

MECHANICAL ENGINEERING: COGNITIVE ROBOTICS
AEROSPACE ENGINEERING: CONTROL AND OPERATIONS

RO57035: Masters Thesis

Author:
Tijn Vennink (5076102)

Mechanical Engineering - CoR Supervisors:
Dr. Ir. Jens Kober

Aerospace Engineering - C&O Supervisors:
Prof. Dr. Guido De Croon
Dr. Ir. Christophe De Wagter
Dr. Ir. Stein Stroobants

Defense Date:
24/07/2026

Asynchronous Event-Based Visual-Inertial Odometry for Flapping-Wing MAVs

Tijn Vennink (5076102)

Department of Mechanical Engineering
Technische Universiteit Delft
Delft, The Netherlands

Abstract— Flapping-Wing Micro Air Vehicles (FWMAVs) operate under severe Size, Weight, and Power (SWaP) constraints, and their high-frequency flapping produces strong oscillatory dynamics that degrade conventional frame-based visual perception. Event cameras are well suited to these conditions, but event-based visual-inertial odometry (EVIO) methods typically rely on short time-window, locally linear motion assumptions that flapping flight breaks: under flapping, event-alignment methods such as plane fitting and contrast maximization recover mostly the high-frequency flapping-induced body rotation rather than the platform’s translation. This work presents the first full EVIO pipeline running entirely on-board a small, high-frequency flapping-wing platform. The front-end deliberately avoids short-window alignment and instead exploits the abundance of flapping-induced events: it detects corner features event-by-event, clusters them into compact spatiotemporal features, and tracks these while discarding the fine oscillatory structure. The resulting feature tracks are fused with inertial measurements in a tightly coupled Extended Kalman Filter. The complete pipeline runs in real time on an on-board microcontroller within a sensing-and-compute payload of roughly 21 g. On flight datasets recorded on-board the FWMAV, the pipeline recovers velocity and attitude during flapping flight, and in a closed-loop flight test the on-board position estimate is accurate enough to hold a stationary hover in the horizontal plane. The same event stream that breaks conventional EVIO assumptions is thus sufficient not only for on-board state estimation, but for closing a position-control loop in flight.

Index Terms—Visual-Inertial Odometry, Event-Based Cameras, Flapping-Wing Micro Air Vehicles, State Estimation, Extended Kalman Filter

I. INTRODUCTION

In aerial robotics, reliable state estimation is essential for autonomous flight without external localization. While larger aerial platforms can often rely on heavier sensor hardware and more capable on-board processors, insect- and bird-scale platforms are constrained by severe Size, Weight, and Power (SWaP) limitations, which restrict both sensor choice and estimator complexity [1], [2]. A particularly compelling family of such platforms are Flapping-Wing Micro Air Vehicles (FWMAVs), which generate thrust through periodic wing motion and control flight through modulation of wing kinematics [3]. The FWMAV platform used in this research is shown in Figure 1.

This Masters Thesis was written under the guidance and supervision of Prof. Dr. Guido De Croon, Dr. Ir. Christophe De Wagter, Dr. Ir. Stein Stroobants, and Dr. Ir. Jens Kober at the Delft University of Technology.

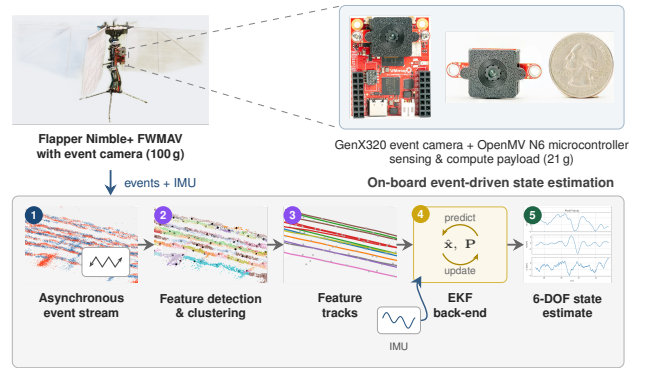


Fig. 1: The GenX320 event camera and OpenMV N6 microcontroller on the Flapper Nimble+ flapping-wing platform [4] capture an asynchronous event stream (1). Events are grouped into spatiotemporal features and tracked over time (2), yielding feature trajectories (3) that can be used for downstream state estimation.

When aerial robots scale down in size, flapping platforms become especially attractive because they exploit favorable aerodynamic scaling at low Reynolds numbers, where rotary-wing propulsion becomes less efficient [2]. At the same time, they introduce an additional perception challenge: the periodic wing actuation generates strong oscillatory body motion and vibrations that disturb the measurements of on-board sensors [2], [5], [6]. As a result, the visual signal available to an estimator is often dominated by flapping-induced motion itself, making conventional motion interpretation difficult. In practice this has kept small FWMAVs largely dependent on external motion capture for state estimation, although on-board stereo vision has enabled autonomous indoor navigation (obstacle avoidance, door traversal, and corridor following) on a 20-gram platform [7].

Event cameras offer a promising alternative to conventional frame-based cameras in this regime. Instead of capturing frames at a fixed rate, each pixel independently emits a timestamped event whenever its local brightness change exceeds a contrast threshold. The resulting asynchronous event stream combines microsecond temporal resolution with high dynamic range and low power consumption [8], [9], properties well matched to the vibrational conditions of flapping-wing flight. Prior studies

on larger FWMAVs confirm this, finding event-based sensing advantageous over frame-based cameras under flapping flight, where the camera's asynchronous response avoids the motion blur that the persistent fast wing motion induces in conventional frame cameras [5], [6], [10].

However, while event cameras have been used for perception and visual guidance on larger flapping-wing platforms, there is, to our knowledge, no prior work demonstrating a full, on-board event-based visual-inertial odometry (EVIO) pipeline on a small, high-frequency FWMAV of this scale. Existing event-based visual and visual-inertial odometry methods are moreover typically built around assumptions of locally smooth or approximately linear motion, which become fragile when the dominant visual motion is mostly rotational, driven by the wingbeat itself rather than by the platform's slower translation.

This work first shows how the flapping-wing event stream breaks these assumptions for existing methods, and then presents a pipeline that nevertheless supports on-board visual-inertial state estimation. Rather than collapsing events into frames or aligning them within short time windows, the proposed pipeline extracts and tracks sparse corner features directly from the asynchronous event stream, and fuses the resulting feature tracks with inertial measurements in a tightly coupled Extended Kalman Filter, within the computational limits of on-board deployment.

The main contribution is not merely the use of an event camera on an FWMAV, but an estimation pipeline shaped around its oscillation-dominated event stream, exploiting the abundance of flapping-induced events rather than their fine oscillatory structure. Beyond accuracy against a recorded reference, we show the estimate is usable in practice: with the system running fully on-board, the horizontal-position estimate is accurate enough to close the control loop for a stationary hover (height held on an external reference). The remainder of the work reviews related work (Section II), motivates the proposed architecture through an analysis (Section III), presents the EVIO pipeline (Section IV), and evaluates its performance on recorded flight data and in an on-board closed-loop hover (Section V).

II. RELATED WORK

A. Event-based sensing on flapping-wing platforms

The use of event cameras on flapping-wing platforms is a young field. Several studies have evaluated event cameras on FWMAVs, reporting dense, high-frequency event activity and robustness advantages over frame cameras [5], [6], [10], and event-based visual guidance of flight maneuvers has been demonstrated in controlled settings [11], with a dedicated event-and-IMU dataset released to support perception research on such flapping-wing platforms [12]. These works, however, target larger FWMAVs, with wing spans above a metre, masses around 500 g, and flapping frequencies near 5 Hz, and their methods have not been tested on smaller platforms (~ 30 cm span, 10–100 g) with much higher flapping frequencies (around 15 Hz).

B. Event-based front-ends

Event-based front-ends for a VIO pipeline fall into two broad groups. The first group extracts and tracks sparse features: arc-based corner detectors such as eFast and ARC* [13], [14], which mark an event as a corner when the most recent events on a small pixel circle around it form a single contiguous angular segment (an “arc”), and event clustering or blob methods [15]–[17] identify compact, trackable structure, which asynchronous feature trackers then follow over time using events [18]–[20]. The closest such front-end to ours has been used on-board an MAV to track objects in the scene [21], not to estimate the platform's own motion under flapping. The second group estimates motion directly by aligning events under a locally constant-motion assumption: plane fitting recovers normal flow in the (x, y, t) volume [22], with real-time FPGA realizations [23], while learning- and contrast-maximization-based methods warp events to maximize the sharpness of the accumulated image [24]–[26]. Across these front-ends, the shared assumption is that motion stays locally coherent within a short temporal window.

C. Lightweight filter-based visual-inertial back-ends

For state estimation on a tight compute budget, tightly coupled filter-based methods are an established lightweight choice: an inertial EKF with IMU-driven prediction and a compact error-state covariance is well suited to embedded use [27], [28], and filter-based visual-inertial odometry such as ROVIO [29] shows that visual features can be fused into such a filter directly, using a bearing-vector and inverse-depth parameterization, without a separate optimization. Event feature tracks have been used for low-latency visual odometry without inertial fusion [30]; event-based VIO adds the IMU, and has been approached three ways. Optimization-based keyframe methods fuse events, often alongside standard frames, in a windowed nonlinear solver [31]; Ultimate SLAM [32] used such a hybrid pipeline for the first autonomous quadrotor flight controlled with an event camera. Filter-based methods fuse event feature tracks in an EKF [33]. Both method types assume the scene motion stays coherent within a short, fixed or flow-adaptive temporal window. Continuous-time formulations [34] instead model the trajectory as a smooth spline, relaxing the windowed assumption, but the batch spline optimization far exceeds the computational resources available on lightweight flapping-wing drones. All target the smoother motion of conventional platforms rather than oscillatory flapping. For the platform dynamics, a low-order flapping-wing model represents the main aerodynamic effect on the body as a linear drag force proportional to body velocity, avoiding a full aerodynamic state [35], and torque coupling between the rotational axes of this platform family has been demonstrated experimentally [36]. Together these provide the building blocks for a lightweight embedded back-end.

Across these three lines of work, event-based sensing on flapping-wing platforms, event-based front-ends, and lightweight filter-based back-ends, no prior work demonstrates an on-board EVIO pipeline on a flapping-wing platform of

this scale (10–100 g, flapping around 15 Hz), the gap this work addresses.

III. FLAPPING-WING EVENT STREAMS AND THEIR IMPLICATIONS FOR PERCEPTION

An analysis (Appendix A) of event streams recorded on the flapping-wing platform [4] used in this work showed that the visual signal is dominated by periodic body rotation rather than by the platform’s slower translational motion. For this analysis, we recorded controlled flapping-flight and non-flapping datasets on the same hardware. The recordings showed consistently higher event rates at higher flapping activity, in line with the hypothesis posed in prior work [6] and with the event-rate observations on larger platforms [5], [10]. It further showed that the dominant temporal component of the event stream on our platform can appear near twice the wingbeat frequency. This amplification is consistent with oscillatory rotational dynamics and higher-order aerodynamic forcing over the wingbeat (a simplified derivation is given in Appendix B). As a result, the event stream is highly non-linear and repeatedly reverses direction, producing rapid viewpoint changes and strongly non-stationary event dynamics even when compared with non-flapping recordings collected with the same hardware and in the same environment.

Motivation for Pipeline Design: Existing event-based front-ends share one core assumption: motion stays locally coherent within a short time window (Section II). To test how well this assumption survives flapping flight, we ran a representative method from each family on our recordings: a local plane-fitting normal-flow estimator, a learning-based contrast-maximization model, and an arc-based corner detector (Appendix A). For the two alignment-based methods, the assumption holds only approximately and only in very short windows: larger windows mix opposing phases of the oscillatory motion, so the dominant rotational component increasingly masks the slower translational motion, while very short windows capture mainly that rotational component. This does not make such methods ineffective in general; rather, their core assumptions become increasingly challenged on small, high-frequency flapping-wing platforms.

Arc-based corner detectors remained operational under flapping flight, but corner yield and the spatial distribution of detections depend strongly on the temporal accumulation window, and this dependence is set by the flapping frequency: windows shorter than the dominant event-oscillation period (near $2f_{\text{flap}}$, Appendix A) disrupt arc formation and yield fewer but more dispersed detections, whereas windows spanning multiple oscillation cycles stabilize the local event structure and increase yield at the expense of stronger spatial clustering. The flapping-flight event stream can thus still be used to extract visual features with an arc-based detector.

Beyond these motion characteristics, our platform is limited by a severe computational budget: the camera and microcontroller form a payload of only 21 g, with on-board processing limited to a microcontroller. A survey of the hardware on which existing event-based methods were demonstrated (Appendix A-Of, Table VI) shows that most target

desktop CPUs or workstation GPUs, and that the few systems flown on-board carry airframes one to two orders of magnitude heavier than the platform used here. This makes most of the existing stack infeasible to deploy directly and, together with the motion findings, favors a lightweight corner-based front-end over heavier event alignment- or learning-based alternatives.

We arrive at the following observations that motivate the proposed design:

- 1) Event cameras are a good sensing match for flapping-wing flight, but no full EVIO pipeline has been demonstrated on a flapping-wing platform of this scale (Section II).
- 2) On the platform’s own data, alignment-based front-ends (plane fitting and contrast maximization) mainly recover the flapping-induced oscillatory rotation, whereas arc-based corner detection remains usable under flapping flight (Appendix A).
- 3) Filter-based visual-inertial methods and low-order flapping-wing dynamics models provide building blocks that fit the on-board computational budget (Section II).

IV. METHODOLOGY

Motivated by the results of the analysis (Section III), this section describes the proposed event-based asynchronous VIO pipeline for FWMAVs. The pipeline has two components (Figure 1): an asynchronous front-end that converts the raw event stream into feature tracks in the image plane, and a tightly coupled EKF back-end in which these tracks drive the measurement update while an IMU-and-drag prediction model drives the propagation, jointly estimating the robot state and the inverse-depth feature states.

A. Asynchronous feature detection and tracking

The front-end operates on individual events in real time, deliberately avoiding long temporal event accumulation: the analysis showed that, under flapping flight, motion coherence is lost once a time window approaches the dominant oscillation period, and that even short windows mainly capture the oscillatory rotation. Unless stated otherwise, the numerical parameter values reported below should for that reason be interpreted as platform-specific operating points chosen empirically to balance feature yield, temporal stability, and on-board compute cost on the platform’s microcontroller (Appendix A-Of), rather than as universally optimal settings. The complete per-stage update equations are given in Appendix F, and front-end parameter values and tuning in Appendix D-A.

1. Redundant-event suppression and timestamp surfaces:

The k -th event in the stream arrives as $e_k = (t_k, x_k, y_k, p_k)$, with timestamp t_k in microseconds, pixel coordinates (x_k, y_k) , and polarity $p_k \in \{+1, -1\}$. The pipeline maintains two polarity-separated timestamp maps, stored as Surfaces of Active Events [22]: a latest-event surface $L_p(x, y)$ updated by every event, and a reference surface $S_p(x, y)$ updated only by accepted events. An event is accepted if

$$t_k > L_p(x_k, y_k) + \kappa \quad \text{or} \quad L_{-p}(x_k, y_k) > L_p(x_k, y_k), \quad (1)$$

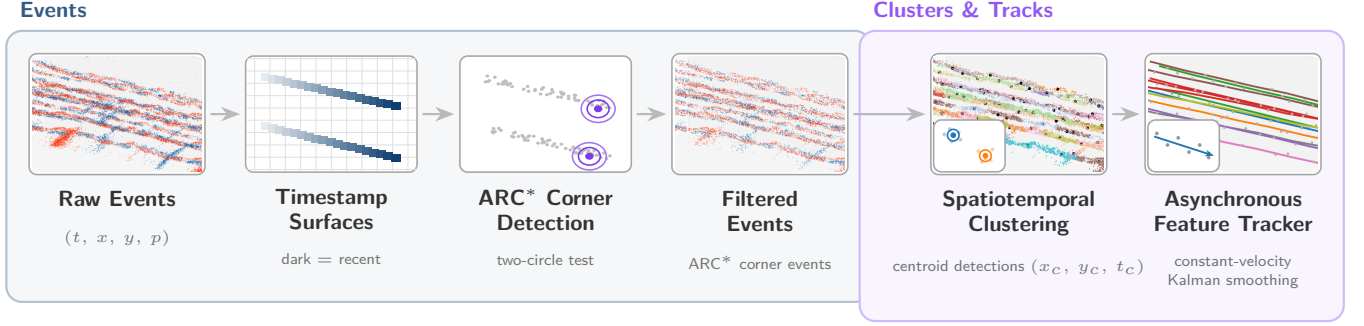


Fig. 2: Asynchronous front-end pipeline, showing raw events, timestamp surfaces, corner detection, clustering, and feature tracking.

a refractory-style criterion in the spirit of per-pixel refractory and background-activity denoising filters for event sensors [37], [38], where the second condition passes genuine sign changes in local contrast. This suppression removes redundant same-polarity spikes and sensor noise, lowering downstream computational load while maintaining the activity required for feature detection and tracking; the threshold κ is tuned for our platform (Appendix D-A).

2. ARC* corner detection: Accepted events are filtered with an asynchronous corner test based on ARC* [14], chosen because it operates per event directly on the timestamp surface and is cheap enough for the on-board microcontroller, while the analysis (Appendix A) showed that this arc-based detector family retains corners under flapping flight. For each event, the test samples the polarity-matched reference surface S_p on two circles centered at the event, following the two-circle construction of eFast [13]. The circles are rings of pixels in the image plane (radii r_s and r_ℓ), read on the timestamp surface: the test reasons about how recently neighbouring pixels last fired, rather than operating in the (x, y, t) volume. When a corner moves across the sensor, the most recently updated pixels on such a ring form one contiguous angular segment, the arc; a straight edge instead updates a larger part of the ring, giving an arc that is too long, and noise yields no contiguous segment at all. The test therefore finds the largest contiguous arc of newest timestamps on each circle, of length $|A_{\text{new}}|$, and classifies the event as a corner if $L_{\min} \leq |A_{\text{new}}| \leq L_{\max}$ holds for this arc or its complement on both circles; requiring both circles reduces false positives from straight edges and noise. The circle radii (r_s , r_ℓ) and arc-length thresholds are tuned for our platform rather than taken from the original detectors (Appendix D).

3. Spatiotemporal clustering: Corner events are aggregated into compact spatiotemporal clusters (event blobs). This stage follows from the analysis (Appendix A): corner detections under flapping remain available but are sparse, dispersed, and short-lived, so we group them with a minimal radius-based setup instead of a richer appearance model, such as the photometric patch templates or learned descriptors used by event feature trackers [18], [39], which would be fragile under fast oscillatory motion and more expensive on-board.

Each cluster c maintains a centroid (x_c, y_c) , last-event

timestamp $t_{c,\text{last}}$, event count n_c , and bounding-box extrema. Clusters not updated within a timeout r_t are deactivated. The timeout is kept short relative to the wingbeat by design: a cluster carries no motion model, so it aggregates a single coherent burst of activity, while bridging the activity gaps between bursts, where the feature reappears displaced, is left to the tracker and its velocity prediction (steps 4–5). For an incoming corner event, only the active clusters with $|x_k - x_c| \leq r_x$ and $|y_k - y_c| \leq r_y$ are considered as candidate clusters; the event is then assigned to the candidate cluster minimizing the normalized elliptical distance $d(c, e_k) = (x_k - x_c)^2 / r_x^2 + (y_k - y_c)^2 / r_y^2$. If several clusters pass this test, they are merged into a single cluster by event-count-weighted centroid averaging and bounding-box union: clusters this close together are assumed to originate from the same feature, and merging them prevents several small, jittery clusters from existing on a single feature. If no cluster satisfies this test, a new cluster is initialized at the event location.

The assigned event updates the centroid with an exponential moving average, $x_c \leftarrow x_c + \alpha(x_k - x_c)$ and likewise for y_c , with smoothing factor α . This damps flapping-induced centroid jitter within a cluster at the event rate, while the constant-velocity tracker (step 4) smooths across cycles at the loop rate, so the two act on different timescales. Once $n_c \geq n_{\min}$, the cluster is passed to the feature tracker as a detection; the measurement is the centroid (x_c, y_c) with the mean event time $t_c = \frac{1}{n_c} \sum_{i=1}^{n_c} t_i$ as its timestamp. The radii, timeout, smoothing factor, and support threshold are tuned for our platform (Appendix D-A).

4. Constant-velocity Kalman model feature tracking: Clustered detections are converted into continuous trajectories by a multi-target Kalman tracker operating in the image plane. Each track carries an image-plane constant-velocity state $\mathbf{x} = [x, y, v_x, v_y]^T$: not a model of the flight dynamics, but a lightweight temporal prior that smooths the non-stationary cluster stream produced by the oscillatory motion. The measurement observes position only, because the instantaneous feature velocity is dominated by the high-frequency flapping component (Section III), which the constant-velocity prior should smooth rather than track. Accordingly, the process noise $\mathbf{Q} = \text{diag}(q, q, 10q, 10q)$ assigns the unobserved velocity states higher uncertainty, with measurement noise $\mathbf{R} = r\mathbf{I}_2$;

the standard prediction and update equations are given in Appendix F-D.

5. Data association: Data association uses gating followed by greedy assignment: all active tracks are predicted to the current detection time, candidate track-detection pairs within a Euclidean gating distance d_g are sorted by distance, and assigned greedily without conflicts. Matched tracks receive the Kalman measurement update, unmatched detections initialize new tracks, and tracks are terminated once the time since the last successful association exceeds a maximum missing interval $t_{\text{miss,max}}$.

A newly initialized track is promoted to a confirmed feature track only once it accumulates n_{birth} successful associations (Appendix F-E). This confirmation suppresses spurious tracks spawned by noise or fragmented clusters, at the cost of a short delay before a stronger feature becomes available to the back-end. For each confirmed track, the tracker outputs the track identifier and image-space position to the back-end.

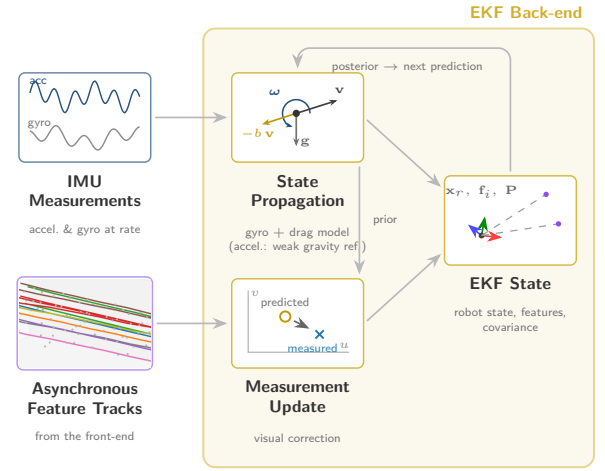


Fig. 3: EKF back-end architecture, showing IMU-driven prediction, measurement update from asynchronous visual observations, and the corrected state output.

B. Extended Kalman Filter Visual-Inertial Odometry

The back-end follows the quadrotor EKF formulation of [27], extended with inverse-depth feature states that are jointly estimated with the robot state and updated by the asynchronous event-based front-end. Because the event stream is dominated by fast oscillatory components (Section III), the IMU-driven prediction provides temporal continuity while the visual tracks serve as the correction source. The full prediction, Jacobians, and update equations are given in Appendix G.

1. State model: The robot part of the state is written as

$$\mathbf{x}_r = [X \ Y \ Z \ P_X \ P_Y \ P_Z \ D_0 \ D_1 \ D_2]^\top. \quad (2)$$

Here (X, Y, Z) denotes robot position in the world frame, (P_X, P_Y, P_Z) the body-frame velocity, and (D_0, D_1, D_2) a small-angle attitude-error state, while the nominal attitude itself is maintained separately as a quaternion. The world, body, and camera frame conventions used throughout are defined in Appendix E. Each tracked feature f_i is represented, following filter-based VIO practice [29], in a bearing-vector and inverse-depth form as

$$\mathbf{f}_i = [b_x \ b_y \ \rho]^\top, \quad (3)$$

where (b_x, b_y) are the tangent-plane coordinates of the feature's world-frame unit bearing $\mathbf{h} \in S^2$ (Appendix G-E) and ρ is its inverse depth. The bearing is handled identically to the attitude: the full unit vector is maintained alongside the state, while the filter estimates a minimal two-dimensional error in its tangent plane. This representation keeps each visual state compact while tolerating uncertain feature range at initialization and the large depth variation encountered during flight. Features are anchored in the world frame rather than to the moving body; the comparison with robocentric anchoring is examined in Section VI. The full EKF state stacks the robot and feature states,

$$\mathbf{x} = [\mathbf{x}_r^\top \ \mathbf{f}_1^\top \ \mathbf{f}_2^\top \ \dots]^\top. \quad (4)$$

2. Covariance structure: The covariance is partitioned into robot, feature, and cross-covariance blocks:

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_{rr} & \mathbf{P}_{rf} \\ \mathbf{P}_{fr} & \mathbf{P}_{ff} \end{bmatrix}. \quad (5)$$

Here, $\mathbf{P}_{rr} \in \mathbb{R}^{9 \times 9}$ is the robot covariance, $\mathbf{P}_{ff}$ contains the per-feature 3×3 covariance blocks, and $\mathbf{P}_{rf}$ stores the robot-feature cross-covariances. Cross-covariances between different features are neglected ($\mathbf{P}_{f_i f_j} = \mathbf{0}$ for $i \neq j$), which reduces memory and compute cost on the embedded platform while preserving the dominant robot-feature coupling required for the update step.

3. Robot motion prediction: To keep the back-end computationally feasible on the low-compute platform, the predictor uses a low-order IMU-driven model that captures the dominant gravity, rotational-coupling, and drag effects without introducing a higher-dimensional aerodynamic state. Because the measured specific force on a flapping platform is dominated by oscillatory thrust and aerodynamic loads on all body axes, no accelerometer channel enters the translational prediction; the body-frame velocities are corrected by the visual updates instead. The linear drag and torque-coupling terms are adapted from prior flapping-wing dynamic models and experimental analyses [35], [36].

Attitude is propagated directly from the gyroscope measurement,

$$\mathbf{q}_{k|k-1} = \exp\left(\frac{dt}{2}\boldsymbol{\omega}_k\right) \otimes \mathbf{q}_{k-1|k-1}, \quad (6)$$

while the position increment in the body frame is modeled as

$$\Delta \mathbf{p}_b = \begin{bmatrix} P_X dt \\ P_Y dt \\ P_Z dt + \frac{1}{2} a_z dt^2 \end{bmatrix}. \quad (7)$$

The world-frame position $\mathbf{r} = [X \ Y \ Z]^\top$ is then propagated as

$$\mathbf{r}_{k|k-1} = \mathbf{r}_{k-1|k-1} + \mathbf{R}_{k-1} \Delta \mathbf{p}_b - \frac{1}{2} g dt^2 \mathbf{e}_z, \quad (8)$$

with $\mathbf{R}_{k-1}$ the body-to-world rotation and $\mathbf{e}_z$ the world up direction.

The body-frame velocity update:

$$\begin{aligned} P_X &\leftarrow P_X + dt(\omega_z P_Y - \omega_y P_Z - gR_{2,0} - b_x P_X + b_x o_x), \\ P_Y &\leftarrow P_Y + dt(-\omega_z P_X + \omega_x P_Z - gR_{2,1} - b_y P_Y + b_y o_y), \\ P_Z &\leftarrow P_Z + dt(a_z + \omega_y P_X - \omega_x P_Y - gR_{2,2} - b_z P_Z + b_z o_z), \end{aligned} \quad (9)$$

with $\mathbf{o} = \boldsymbol{\omega} \times \ell_{\text{CoP}}$ denoting the center-of-pressure offset contribution and (b_x, b_y, b_z) the linear drag coefficients. The $-gR_{2,i}$ terms project the world-frame gravity direction onto the body axes (Appendix E). The thrust-axis specific force a_z is not taken from the accelerometer but fixed to g under the near-hover assumption that thrust balances gravity; the measured- a_z variant is examined in Appendix D, and the complete in-flight predictor is detailed in Appendix G-B. The accelerometer serves only as a weakly weighted roll-and-pitch reference once per cycle (Appendix G-H).

The gyroscope is low-pass filtered before it enters the prediction. The wing-beat and structural vibration occur at much higher frequencies than the body-rotation dynamics the filter tracks (Section III), and propagating the raw rate would feed this vibration into the integrated attitude and, through the rotational-coupling terms above, into the predicted velocity; the filtered rate is therefore used as $\boldsymbol{\omega}_k$ throughout the prediction. The filter realization and cutoff are given in Appendix G-B.

4. Linearization and covariance prediction: Linearization of the motion model yields the state-transition Jacobian

$$\mathbf{A}_k = \frac{\partial f}{\partial \mathbf{x}_r}. \quad (10)$$

The robot covariance propagates as

$$\mathbf{P}_{rr,k|k-1} = \mathbf{A}_k \mathbf{P}_{rr,k-1|k-1} \mathbf{A}_k^\top + \mathbf{Q}_k. \quad (11)$$

Here dt is the measured time elapsed during one prediction cycle; $\mathbf{Q}_k$ is built from continuous-time noise spectral densities, the same process-noise parameters remain valid across update rates (Appendix G-I). The block structure of $\mathbf{A}_k$ and the process-noise construction are given in Appendix G-C and Appendix G-D. After each correction, the covariance handling for the attitude-error state follows the covariance-correction treatment for attitude-based Kalman filters proposed in [28], adapted to our world-frame state and inverse-depth feature parameterization.

5. Feature propagation: Tracked visual features are assumed static over the prediction interval, so the feature states and their covariance blocks remain unchanged, $\hat{\mathbf{f}}_{i,k|k-1} = \hat{\mathbf{f}}_{i,k-1|k-1}$; only the robot-feature cross-covariances propagate with the robot dynamics, $\mathbf{P}_{rf,k|k-1} = \mathbf{A}_k \mathbf{P}_{rf,k-1|k-1}$.

6. Measurement update: A confirmed front-end track not yet in the filter is inserted as a new feature: the track's current undistorted observation is back-projected at the initial inverse depth and stored as a world-frame unit bearing, so the bearing is taken directly from the measurement and the inverse depth is the only tuned quantity at initialization. The inverse depth is set to the median inverse depth of the converged features in the map, so a new feature inherits the scene depth already established by the filter; a fixed prior serves as fallback when fewer than three qualify (Appendix G-F).

Each tracked feature then provides an image-plane measurement

$$\mathbf{z}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} = h(\mathbf{x}_r, \mathbf{f}_i) + \mathbf{n}_i, \quad (12)$$

with noise covariance $\mathbf{R}_i$. Tracked pixel coordinates are first converted to undistorted normalized image coordinates $\tilde{\mathbf{z}}_i$ using the calibrated camera model, so intrinsics and lens distortion are handled once, outside the filter. The residual $\mathbf{r}_i = \tilde{\mathbf{z}}_i - \hat{\mathbf{z}}_i$ compares the observed track position with the predicted projection of the feature into the current camera view; because this projection depends on both the robot pose and the feature state, through the combined Jacobian $\mathbf{H}_i = [\mathbf{H}_{r,i} \ \mathbf{H}_{f,i}]$, each observation corrects the robot and feature estimates jointly. The innovation covariance is $\mathbf{S}_i = \mathbf{H}_i \mathbf{P} \mathbf{H}_i^\top + \mathbf{R}_i$.

This joint correction also makes the filter vulnerable to outliers: the front-end associates features by image position alone, without an appearance template, and occasionally produces mismatched or short-lived tracks under flapping motion, and a single large incorrect correction would propagate into all coupled states. Each observation is therefore screened by a Mahalanobis gate, $d_i^2 = \mathbf{r}_i^\top \mathbf{S}_i^{-1} \mathbf{r}_i \leq \gamma$; a rejected feature remains in the map and may be re-observed later. Accepted observations are applied with the standard Kalman gain and covariance update; the χ^2_2 -based gate threshold, the warm-up interval during which the gate is disabled, and the partitioned robot and feature gain expressions used in our implementation are given in Appendix G-G.

V. RESULTS

This section reports the experimental evaluation: the flight-data collection, the front-end feature-tracking statistics, the back-end state-estimation accuracy against a motion-capture reference, and the on-board experiments, including a closed-loop hover flown live on the estimate.

A. Data collection and setup

Data was collected using a Flapper Nimble+ platform [4] equipped with a GenX320 event-based camera and an OpenMV N6 (STM32N657) microcontroller. The GenX320 is a lightweight (~ 3 g) event camera optimized for low-power applications. During the data-collection flights, the N6 ran the asynchronous front-end and logged its feature-track output together with the IMU stream. Both are shown in Figure 1. A small bracket (~ 1 g) mounts the parts to the platform, for a sensing-and-compute payload of roughly 21 g; only the raw-event recordings of the preliminary analysis used a different, comparable board (OpenMV H7 Plus, Appendix A).

The flight experiments were conducted inside the CyberZoo at the TU Delft, which is equipped with a motion capture system providing accurate ground truth for evaluation. The FWMAV was manually controlled to fly facing a wall with visual features in white on a black curtain background, as shown in Figure 4. The flight conditions were kept controlled, with speeds below approximately 1 m/s, limited maneuvering, and the wall-facing orientation maintained throughout to ensure sufficient visual features for tracking. The recorded trajectories

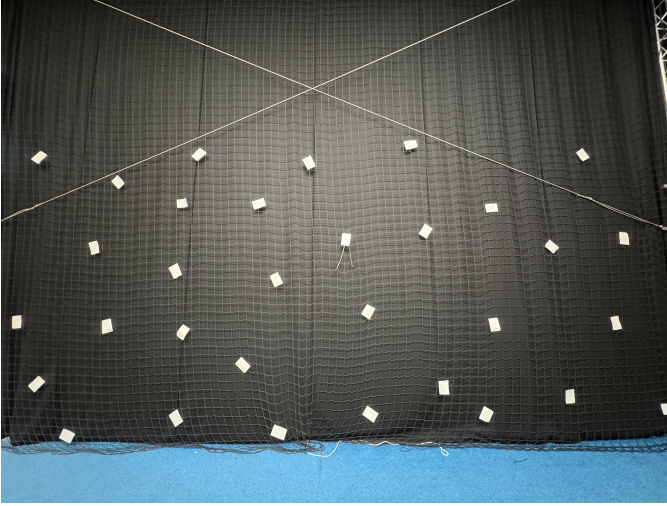


Fig. 4: Visual features on the curtain background used for tracking, inside the CyberZoo at the TU Delft.

include motion in all directions, albeit within these constraints. The datasets are available for reproducibility; details are provided in Appendix K.

The evaluation presented in this section is based on 16 short flights: 13 tuning flights (A–M), on which the filter parameters were optimized, and 3 held-out flights (N–P) reported separately in the generalization check below. After discarding liftoff, the RMSE values of Table I are computed over roughly 13.3 minutes of tracked flight from the tuning flights, given as the mean and standard deviation across flights; a per-flight breakdown of all 16 flights is provided in Appendix J, Table X.

B. Feature tracking performance

The front-end is analyzed directly from its on-board output: all quantities reported in this subsection were captured on-board during the flights, using the front-end settings listed in Appendix D-A, and are pooled over the same set of flights used for the back-end evaluation. Figure 5 shows the lifetime distribution of confirmed feature tracks, pooled over all flights, where lifetime is the elapsed time between a track’s first and last observation. Only confirmed tracks are counted: in the data-association stage of the front-end (Section IV, step 5), a newly initialized track is promoted to a confirmed feature track (and thereby made available to the EKF) only after $n_{\text{birth}} = 96$ successful associations (Table VII). The statistics describe the features actually used by the back-end and exclude short-lived tracks. The 1302 confirmed tracks have a median lifetime of 4.68 s (mean 7.61 s), with a strongly right-skewed distribution. These lifetimes span many wingbeats despite the oscillatory motion, supplying the back-end with temporally stable measurements.

Figure 6 shows the image-plane distributions of all feature detections, pooled over the same flights (398,992 detections) and shown both per flight and as the across-flight mean. The two axes behave differently: the horizontal (x) distribution is broad and relatively flat across the sensor, whereas the vertical (y) distribution is sharply peaked near the principal point (c_x, c_y) $\approx$

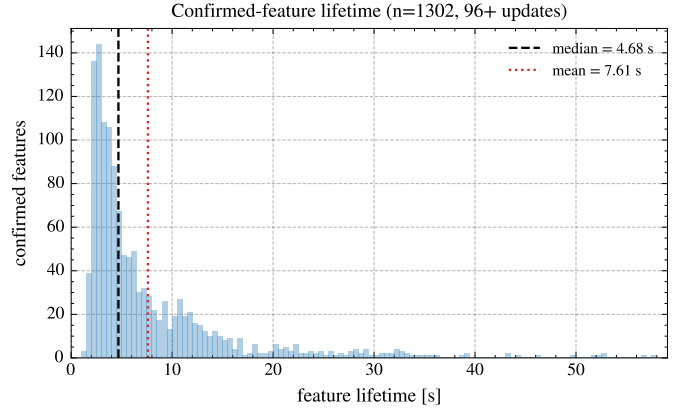


Fig. 5: Lifetime histogram of the $n = 1302$ confirmed feature tracks ($n_{\text{birth}} = 96$ successful associations or more), pooled over all flights and captured on-board. The dashed and dotted lines mark the median (4.68 s) and mean (7.61 s).

(160, 153) and tapers toward the top and bottom image bands. Coverage is broad horizontally but concentrated vertically, and the per-flight curves cluster closely around the mean, indicating that this pattern is consistent across flights rather than driven by any single recording. This distribution reflects the planar feature wall and the limited vertical spread of trackable structure in the current setup: features spread along the horizontal extent of the wall but are spread less vertically. This is a limitation of the test setup rather than of the method: the visual features covered the full width of the curtain but could only be mounted up to a limited height.

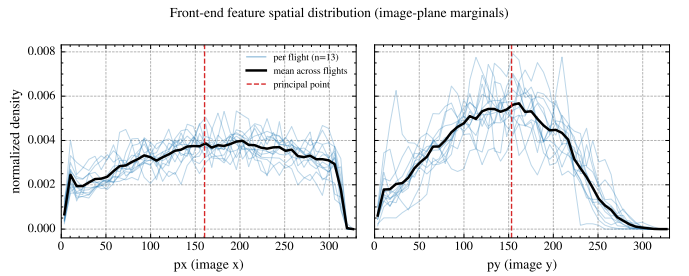


Fig. 6: Image-plane distributions of all feature detections along image x and y (normalized density), mean over all flights (398,992 detections, captured on-board). Light curves show the individual flights, the bold curve the across-flight mean, and the dashed red line the principal point (c_x, c_y) $\approx$ (160, 153). The horizontal distribution is broad while the vertical distribution is concentrated around the image center.

C. State estimation performance

The back-end accuracy is evaluated by replaying the on-board-recorded front-end feature tracks and IMU data through the back-end offline, for two reasons: the replay makes the evaluation repeatable, allowing the noise parameters to be swept and optimized off-board over the 13 tuning flights (Appendix C), and logging the raw event stream required for analysis while simultaneously running the full pipeline is not

feasible on the microcontroller, as writing to the SD card consumes too large a share of the available compute. Since event cameras only generate measurements in response to brightness changes, only limited visual information is available during stationary conditions and immediately after takeoff. For that reason, evaluation was initialized once the FWMAV reached a stable hover shortly after takeoff. Each estimated trajectory is time-synchronized to the motion-capture reference by hand and then aligned to it by a 4-DOF transform, estimating a single yaw rotation about the gravity axis together with a 3-D translation in closed form (least squares) over the evaluation window. This alignment is chosen because the global yaw and position are the only unobservable degrees of freedom of the gravity-aided estimator. Roll, pitch, and scale are deliberately left unaligned, so that residual error is reported rather than absorbed by the fit. Scale still carries a fixed prior through the feature initialization (Section IV): the fallback inverse depth of 0.28 (≈ 3.6 m; Table VII) was fixed from the known test-area geometry (1.5–4 m to the wall), not optimized; under weak near-hover parallax it indirectly sets part of the metric scale, and its contribution is isolated in the depth-prior ablations (Appendix D): accuracy varies little over the geometrically plausible fallback range, and is likewise insensitive to the initialization tightness for over an order of magnitude around the adopted value. Throughout the results, forward, lateral, and vertical (up) refer to the world-frame X, Y, and Z axes, and roll, pitch, and yaw to rotations about the body forward, lateral, and vertical axes; when a quantity is expressed in the body frame we say so explicitly, and the full frame conventions are defined in Appendix E.

Across the evaluated flights, the proposed method tracks the FWMAV’s attitude and world-frame velocity reasonably well, while the estimated position accumulates drift: a mean ATE of 0.75 m over flights averaging 61 s of evaluated flight. Table I reports the RMSE as the mean $\pm$ standard deviation over the evaluated flights, computed after the 4-DOF alignment described above. On average, the position error is comparable across all three axes (0.45, 0.42, and 0.38 m for X, Y, and Z), with no single direction standing out. The velocity error is similarly balanced (0.23–0.31 m/s per axis). For attitude, roll is estimated most accurately (7.1°), followed by pitch (9.1°), while yaw carries the largest error (14.7°) and the largest spread across flights. Whether this accuracy is sufficient to close the loop for a stable hover is examined in the on-board experiments below.

Because the filter parameters were tuned on these same 13 flights (Appendix C), the errors above are in-sample. To test whether the tuning generalizes, we additionally evaluated three flights that were never part of the parameter search, recorded in the same environment. All three were evaluated once, with the tuned parameters unchanged; Table II compares the two sets. The held-out position error is modestly higher (0.88 versus 0.75 m ATE), while velocity, roll, and yaw are essentially unchanged; only pitch is somewhat higher (12.2° versus 9.1° ; Appendix J-C). No held-out flight diverged, suggesting that the tuned parameters transfer to unseen flights rather than being overfit to the tuning set.

The quality of the velocity and attitude estimates is

TABLE I: State-estimation RMSE, reported as mean $\pm$ standard deviation across the 13 tuning flights, after the 4-DOF alignment. Position and velocity are per world-frame axis (X forward, Y lateral, Z vertical; Appendix E).

Metric	RMSE
Position X [m]	0.45 ± 0.19
Position Y [m]	0.42 ± 0.22
Position Z [m]	0.38 ± 0.14
Velocity X [m/s]	0.28 ± 0.07
Velocity Y [m/s]	0.31 ± 0.09
Velocity Z [m/s]	0.23 ± 0.08
Roll [deg]	7.1 ± 2.4
Pitch [deg]	9.1 ± 3.9
Yaw [deg]	14.7 ± 7.4
ATE [m]	0.75 ± 0.25
ATE, median [IQR] [m]	$0.78 [0.62, 0.88]$

TABLE II: Generalization check: mean RMSE on the 13 tuning flights versus 3 flights held out of the parameter tuning (same environment). Per-flight values are in Appendix J-C.

Split	ATE [m]	Vel. [m/s]	Roll [$^\circ$]	Pitch [$^\circ$]	Yaw [$^\circ$]
Tuning ($N=13$)	0.75	0.48	7.1	9.1	14.7
Held-out ($N=3$)	0.88	0.50	6.8	12.2	15.3

illustrated in Figures 7 and 8 for a representative flight segment (flight F, a tuning flight). The world-frame velocity estimate follows the motion-capture reference across all three axes. The body attitude is captured well for roll and pitch, whereas the yaw estimate shows a visible bias with respect to the reference, consistent with its larger RMSE in Table I. Notably, the motion-capture reference itself oscillates strongly at high frequency, most visibly in yaw and in the lateral and vertical velocity. This is independent, motion-capture-based confirmation that flapping induces genuine high-frequency body motion dominated by the yaw axis, mirroring the yaw-dominated event oscillation identified in the analysis (Appendix A). The system intentionally smooths this flapping-frequency oscillation in the feature-tracking stage (the front-end smooths the clustered detections and supplies position measurements only), so the estimate recovers the lower-frequency FWMAV motion by design.

The residual position drift varies between datasets, and the axis in which it is largest is not consistent from flight to flight (Appendix J, Table X). Figure 10 shows the world-frame position for flight F: the estimate follows the reference on all three axes with slowly varying offsets, and for this flight the drift is largest in the vertical axis. Per-flight RMSE values for all flights, together with the velocity and attitude of a higher-drift flight (P), are provided in Appendix J.

To characterize how the position error accumulates, Figure 9 reports the position drift over trajectory segments of increasing duration, pooled over the $N = 13$ tuning flights. Here the drift of a segment is the position error accumulated across it, evaluated over all segments of a given duration in every flight. The median drift grows from 0.06 m over a 1 s segment to 0.38 m over 10 s and 0.80 m over 40 s. The growth is sub-linear, with the per-second drift rate falling from ~ 5.8 to

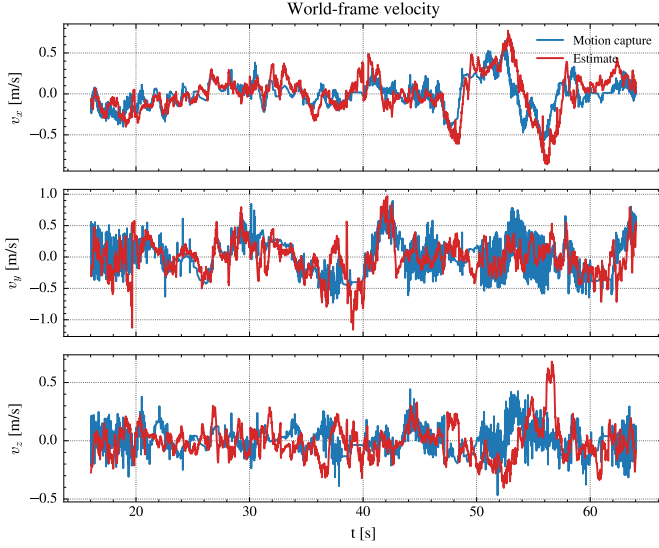


Fig. 7: World-frame velocity estimate over a representative flight segment (flight F), compared to the motion-capture reference (estimate in red, motion capture in blue).

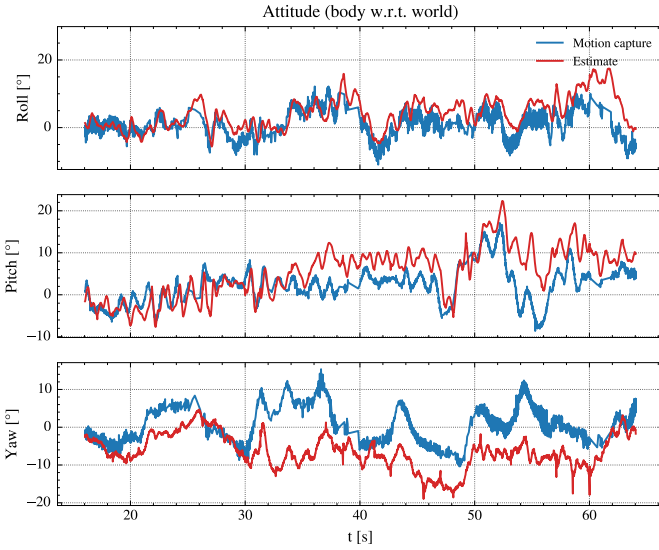


Fig. 8: Body attitude with respect to the world frame over a representative flight segment (flight F), compared to the motion-capture reference (estimate in red, motion capture in blue).

~ 2.0 cm/s as the segment lengthens. Up to ~ 20 s the drift is nearly isotropic, with per-axis medians within roughly 15% of each other; over longer segments the lateral axis grows fastest (0.47 m at 40 s, versus 0.38 forward and 0.33 up). For attitude, the bottom panel reports the within-flight drift: the segment-mean of the attitude error after zeroing each flight's error at its start. Its medians stay within roughly $\pm 3^\circ$ at every duration: roll is centered near zero, yaw stays within about -1.5° , and pitch carries a persistent positive offset of up to $\sim 3^\circ$. The attitude error is thus dominated by its slowly varying level (Table I) rather than by error accumulating within a flight; the distribution behind these medians is shown in Appendix J-B.

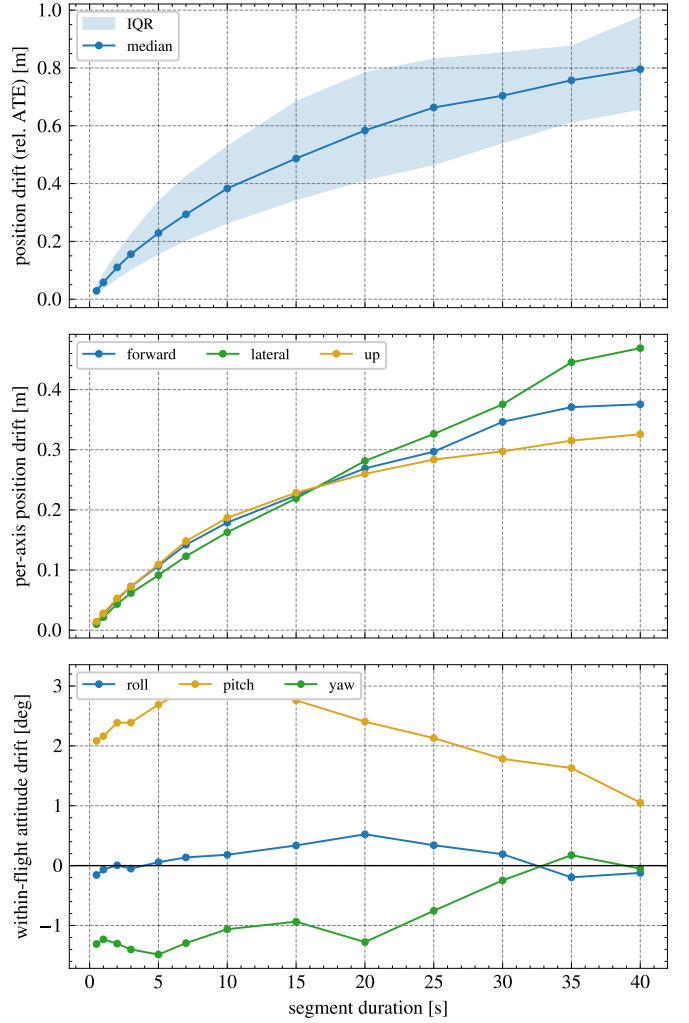


Fig. 9: Drift over trajectory segments of increasing duration, pooled over the $N = 13$ tuning flights. Top: median and interquartile range of the position error accumulated over a segment. Middle: per-axis median position drift, nearly isotropic up to ~ 20 s, with the lateral axis growing fastest over longer segments. Bottom: median signed within-flight attitude drift (segment-mean of the error, zeroed at flight start); all medians stay within roughly $\pm 3^\circ$, with a persistent positive pitch offset.

D. On-board experiments

On-board hover: To test whether the estimated state is accurate enough to fly on, and not only accurate against a reference, the complete pipeline was run on-board in closed loop during a stationary hover. The FWMAV runs its own flight firmware and state-estimation EKF, which already accepts external position measurements [4]. Because tuning the on-board controller to fly directly on the VIO estimate proved difficult, we kept the integration minimal: rather than replacing the platform's stack, the on-board VIO EKF's horizontal position was injected as an x/y position measurement into the firmware EKF, and the platform was flown with the stock Flapper controller gains and firmware-EKF settings unchanged (Appendix K); a native integration is an identified

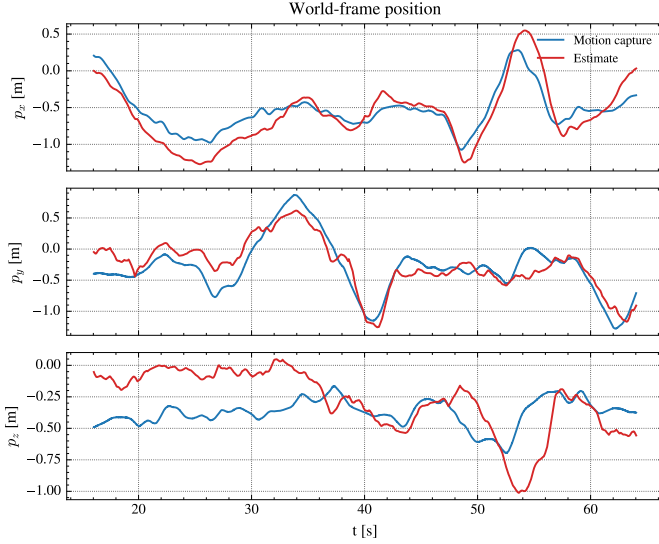


Fig. 10: World-frame position estimate over a representative flight segment (flight F), compared to the motion-capture reference (estimate in red, motion capture in blue).

future direction (Section VII). The integration was limited to the horizontal axes; the height (z) was held on an external motion-capture reference and the attitude left to the platform’s own on-board stack. The FWMAV then held a commanded hover setpoint with its horizontal position controlled by the on-board event-based VIO estimate. For the horizontal axes this closes the gap left by the split evaluation above: the back-end accuracy (Table I) and the on-board real-time feasibility (below) are shown to hold at the same time, live, on the same hardware.

Figure 11 shows the x , y , and z position over a 60-second hover, comparing the on-board VIO estimate with the OptiTrack reference. In the horizontal plane the platform held the setpoint using only the VIO estimate: the ground-truth position stayed within 0.25 m RMS of the setpoint (per-axis standard deviation 0.17 m in x and 0.19 m in y), at a mean of 13 tracked features and an on-board loop rate of 68 Hz. As expected, the bottom panel shows the VIO’s own height estimate drifting freely while the true height, held on the external reference, stays flat; height estimation was not part of the closed loop. Table III summarizes the run. A video of the hover is available at <https://github.com/TijnVennink/onboard-evio-fwma>.

TABLE III: On-board closed-loop hover summary. The horizontal (x, y) loop was closed on the on-board VIO estimate; the height (z) was held on an external OptiTrack reference and the attitude by the platform’s own stack.

Duration [sec]	Pos. std $x/y/z$ [m]	Max. error [m]	Loop [Hz]
60.0	0.17/0.19/0.01	0.53	68

On-board inference: The microcontroller is clocked at 800 MHz with 4.2 MB of on-chip SRAM. The compact front-end, filter state, covariance, and event timestamp surfaces all fit within this on-chip SRAM, so the pipeline is feasible on-board in both compute and memory.

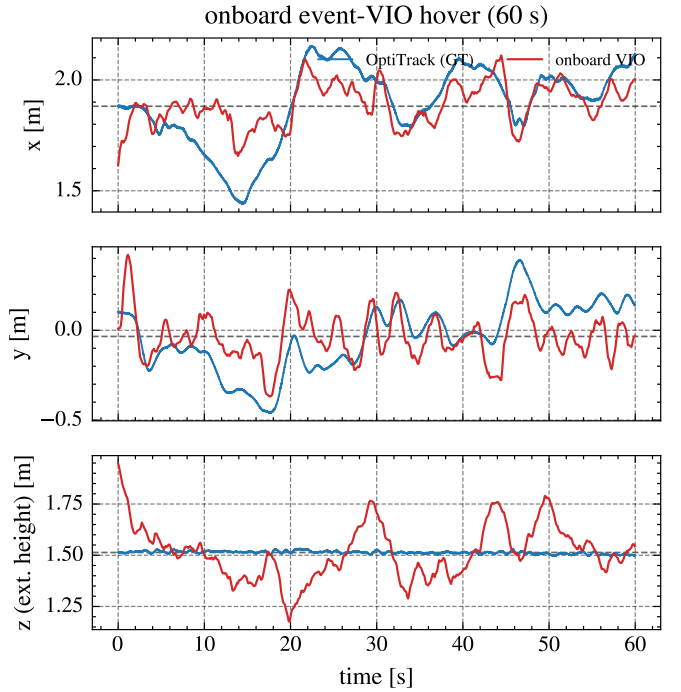


Fig. 11: Position during a 60-second on-board closed-loop hover: on-board event-based VIO estimate (red) versus the OptiTrack reference (blue), with the commanded setpoint dashed. The horizontal axes (x, y) were controlled by the VIO estimate; the height (z) was held on an external OptiTrack reference, so the VIO’s own z estimate (bottom panel) drifts freely while the true height stays flat.

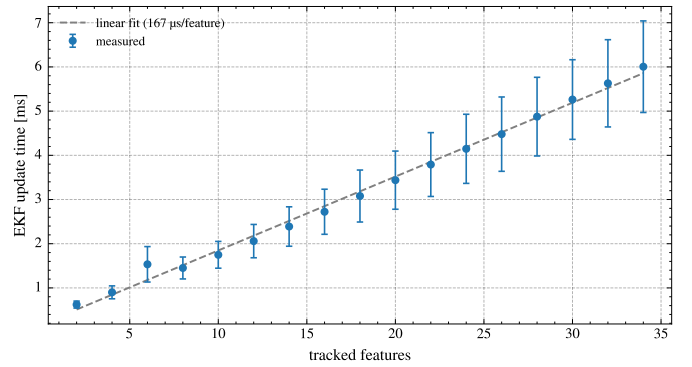


Fig. 12: EKF update time on the OpenMV N6 (STM32N657, Cortex-M55 @ 800 MHz) as a function of the number of tracked features. Markers show the measured mean over 2000 repetitions with ± 1 standard-deviation error bars; the dashed line is a linear fit.

The EKF update cost depends strongly on the number of tracked features. Measured on-board in isolation, it is dominated by the per-feature measurement updates and grows approximately linearly over the tested range, from 0.62 ms at 2 features to 6.0 ms at 34 ($\approx 167 \mu\text{s}$ per feature; Figure 12). The maximum feature count thus acts as a direct control parameter for worst-case update latency: capping the map at N features bounds the EKF update to roughly $N \times 167 \mu\text{s}$.

Computational profile: To identify the dominant computational costs, the complete VIO loop was profiled on-board. Table IV reports the per-stage time when operating with 8 features, averaged over steady-state profiling windows. At this operating point the pipeline runs at approximately 89 Hz (11.3 ms per cycle) and is clearly dominated by the ARC corner detection stage, which alone accounts for roughly 61% of the runtime. Spatiotemporal clustering, feature tracking, I/O, and the remaining overhead each contribute about 6–8%, and the EKF update accounts for 11.6% (1.31 ms). This confirms that front-end event processing, rather than the filter itself, is the primary computational bottleneck of the current implementation, indicating that corner detection is the most promising target for further optimization.

An important caveat is that the ARC corner detection cost scales primarily with the raw event rate produced by the sensor, not with the number of tracked features. The profile in Table IV for that reason holds for the event rates encountered in these recordings; scenes or maneuvers that generate substantially fewer events would decrease the front-end share and increase the achievable cycle rate.

TABLE IV: Measured computational cost breakdown of the proposed pipeline, profiled on-board (OpenMV N6) over the full VIO loop while tracking 8 features (~ 89 Hz). Values are the mean per-cycle time and runtime share across steady-state profiling windows.

Pipeline stage	Mean time [ms]	Runtime share
ARC corner detection	6.92	61.2%
Spatiotemporal clustering	0.75	6.7%
Feature tracking	0.65	5.8%
EKF update	1.31	11.6%
I/O	0.91	8.1%
Other overhead	0.75	6.6%
Total	11.29	100%

E. Ablation studies

To show that the reported accuracy follows from the proposed design rather than from any single component in isolation, and to justify the main design choices made in Section IV, we ablate the pipeline. The headline ablation, disabling the visual updates so the estimator runs on inertial prediction alone, is reported here because it directly isolates the contribution of the event-based front-end. The remaining front-end and back-end ablations quantify the sensitivity to individual parameters and justify the specific values used.

Visual updates versus inertial-only prediction: The visual measurement updates are the mechanism by which the event-based front-end corrects the IMU-driven prediction. To isolate this contribution, we compare the full pipeline against an inertial-only variant in which the visual update is disabled while the prediction step is left untouched (Section IV). Both estimates are aligned to the motion-capture reference over the evaluation window (the 4-DOF alignment of Section V), so the reported inertial-only error is the aligned residual rather than raw integration drift from the origin. Figure 13 and Table V report a representative flight.

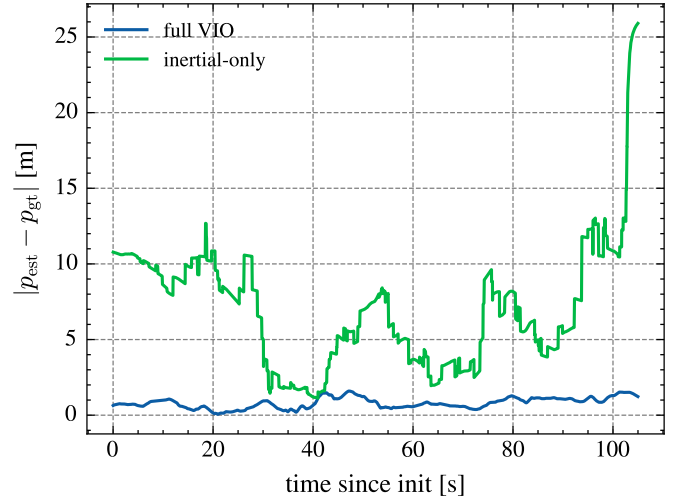


Fig. 13: Absolute position error $|p_{\text{est}} - p_{\text{gt}}|$ over time for the full pipeline versus the inertial-only variant on representative flight I

TABLE V: Effect of the visual updates: ATE, velocity, and per-axis attitude RMSE for the inertial-only variant versus the full pipeline on representative flight I. The full-VIO column coincides with flight I's row in Table X.

Metric	Inertial-only	Full VIO
ATE RMSE [m]	7.95	0.88
Velocity RMSE [m/s]	8.58	0.42
Roll RMSE [deg]	48.3	6.2
Pitch RMSE [deg]	34.9	10.3
Yaw RMSE [deg]	86.3	15.0

Disabling the visual updates degrades every metric severely: the ATE rises from 0.88 to 7.95 m, the velocity RMSE from 0.42 to 8.58 m/s, and the roll/pitch/yaw RMSE from $6.2^\circ/10.3^\circ/15.0^\circ$ to $48.3^\circ/34.9^\circ/86.3^\circ$ (Table V). Figure 13 shows the mechanism: the full pipeline holds the position error around a metre or below for the entire flight, whereas the inertial-only estimate wanders between roughly three and thirteen metres and diverges to about 25 m by the end. The largest inertial-only attitude error is in yaw (86°), reflecting that yaw is unobservable from the inertial measurements alone; the gravity reference constrains only roll and pitch (Section IV), so without the visual updates the heading drifts freely and roll and pitch degrade as well. Taken together, this confirms that the visual measurements supplied by the event-based front-end, rather than the low-order inertial model, are the dominant source of the system's accuracy.

Summary of the design-choice ablations: The remaining studies justify individual front-end and back-end design choices; each is reported in full in Appendix D. On the front-end, every hand-set parameter behaves according to the mechanism it is meant to control, no value sits at a poor operating point, and the governing scales are set by the platform rather than the algorithm: the timescales follow the flapping frequency (clustering operates within a wingbeat, track association bridges across wingbeats) and the spatial scale follows scene depth

and feature size. In brief:

- **ARC corner filtering.** Enabling the corner test yields fewer but longer-lived, less jittery tracks (+26% median lifetime on the test recording), by discarding the edge and flat-region events that seed unstable clusters.
- **Redundant-event suppression.** The flown threshold $\kappa = 1$ ms removes only 0.2% of events; removal becomes meaningful near the event-oscillation period (19% at 10 ms) with no measured quality loss, making κ a conservative throughput knob with compute head-room.
- **Spatiotemporal clustering.** The cluster age-out shows a knee exactly at the event-oscillation period $1/(2f_{\text{flap}})$, and the spatial radius is the front-end's primary scale knob, with an optimum set by scene depth and feature size rather than by the algorithm; both motivate flap- and depth-aware adaptation.
- **Tracker association and lifetime.** The association gate (20 px) and track-death timeout (80 ms, one wing-beat) both sit at their principled operating points; the timeout is bounded by the flapping period, beyond which tracks merely coast without measurements.
- **Depth-prior fallback.** Sweeping the fallback inverse depth shows accuracy is nearly constant across the geometrically plausible depth range, degrading steeply only when the prior places features far beyond the true scene; the flown value lies near the edge of that range rather than at its most accurate point, consistent with a prior fixed from geometry rather than optimized.
- **Depth-initialization tightness.** Scaling the initial inverse-depth uncertainty from $0.1\times$ to $1000\times$ the adopted value leaves accuracy unchanged for over an order of magnitude around the operating point; only near-uninformative initializations degrade the estimate, so the reported accuracy does not hinge on this tuned value.
- **Thrust input to the predictor.** Inserting the measured body- z (thrust) acceleration into the prediction degrades vertical-velocity accuracy on 12 of the 13 tuning flights (mean v_z RMSE $0.23 \rightarrow 0.37$ m/s), supporting the thrust-free configuration used in this work.

VI. DISCUSSION

A. Position drift

A clear pattern in the results is a split between what the filter tracks well and what it does not. World-frame velocity follows the reference on all three axes, with comparable per-axis RMSE (0.23–0.31 m/s; Table I). This is notable because the predictor inserts no direct accelerometer input on any axis (Section IV): the velocities are propagated through gyroscope-driven attitude, gravity projection, rotational coupling, and drag, and corrected only by the visual updates. Body attitude is likewise captured, though not uniformly: roll and pitch are estimated to 7.1° and 9.1° , whereas yaw is markedly weaker (14.7°) and shows a persistent bias with respect to the reference (Table I, Figure 8). In contrast, absolute position drifts: the per-axis position RMSE is comparable across X, Y, and Z, no single direction dominates, and the axis carrying the largest drift differs between flight

recordings (Table X). This is to be expected: without loop closure or an absolute positional reference, global position is unobservable, so the estimate accumulates slow drift relative to the reference origin. The segment-based drift analysis (Figure 9) shows that this accumulation is sub-linear in time, with the per-second drift rate falling from ~ 5.8 cm/s over 1 s segments to ~ 2.0 cm/s over 40 s: consistent with error growth limited by the visual corrections rather than with free inertial integration, whose error grows far faster once the visual updates are disabled (Table V). Up to ~ 20 s the drift is also nearly isotropic; under weak parallax one might expect the weakly observed depth direction to dominate, but the residual scale error spreads over all axes. Over longer segments the lateral axis grows fastest, and the attitude panel of the same figure points to the mechanism. The within-flight attitude drift medians stay within $\pm 3^\circ$, so the attitude error itself does not accumulate; but yaw, unobservable from gravity and only weakly constrained by the planar scene (see below), shows by far the widest spread of offsets across flight segments (standard deviation $\sim 9.9^\circ$ over 10 s windows; Appendix J-B). A heading offset that persists over the length of a segment misprojects the estimated motion sideways with one sign, so the lateral position error keeps integrating even though the yaw error itself remains bounded.

B. Lower frequency by design

A natural objection to the smooth velocity and attitude estimates is that they simply miss part of the true motion. The motion-capture reference, a source independent of the pipeline, shows that the high-frequency motion is real: it oscillates strongly, most visibly in yaw (Figures 7 and 8), reproducing the event-stream oscillation found in the analysis (Appendix A, Figure 17). The estimator smooths this band by design: the front-end clusters and smooths event structure, and the visual measurements passed to the filter are positions only, not the velocities that carry the flapping component. The smoother estimates are thus the lower-frequency platform movement recovered by design, not missed tracking; consistently, the feature tracks themselves persist across many wingbeats (median lifetime about 4.7 s, Figure 5). The pipeline's central design choice, spending the abundance of flapping-induced events on sparse trackable features while discarding the oscillatory structure, holds up during flapping flight.

C. On-board feasibility and computational cost

The pipeline runs at ~ 89 Hz on-board, with the full state, covariance, and event surfaces kept in on-chip SRAM. This provides real-time event-based VIO on a microcontroller in this weight class, which has not been shown before on a flapping-wing platform. A useful detail is where the compute goes. ARC corner detection consumes about 61% of each cycle, while the EKF update is cheap (1.31 ms, 11.6%) and scales at roughly $167 \mu\text{s}$ per feature, making the feature cap a solid latency parameter (Table IV, Figure 12). The compute bottleneck is the front-end, and the back-end thus has computational headroom if the front-end can be made lighter. And since ARC corner check cost is driven mainly by the raw event count rather than the number of tracked features, both

the runtime and the accuracy gains could be most impactful at the front-end.

D. Relation to filter-based VIO

Architecturally the back-end is similar to the filter-based VIO discussed in the related work section, sharing IMU-driven prediction and inverse-depth bearing features. The main difference is the front-end: photometric patch alignment assumes a locally coherent appearance, which the event stream under flapping flight does not provide, and is replaced here by arc-based corners and cluster tracking. The correction available to the filter is accordingly weaker in principle: each feature contributes only an image-plane position, with no photometric patch or appearance information behind it. A richer measurement, such as matching the local event structure around a feature, is possible in principle but was deliberately not used: the analysis showed that this structure loses coherence under flapping (Section III), and incorporating it in a flapping-aware form is left as a future direction (Section VII). That the reduced, position-only measurement still corrects the state during flapping flight shows that the geometric information in the sparse tracks suffices in this regime.

E. World versus robocentric feature anchoring

A second architectural difference lies in the feature anchoring. Some filter-based methods anchor features robocentrically [29], which keeps the unobservable global states out of the filter and thereby avoids the over-confidence that linearization can create along unobservable directions [40]. That consistency benefit is, in principle, most valuable exactly when observability is weak, so it also speaks for robocentric anchoring in the low-parallax regime flown here. We nevertheless anchor features in the world frame, for two practical reasons: a moving anchor recomputes every feature at each prediction step from the accumulated, noisy ego-motion, which near hover, with little parallax to correct those transforms, mainly injects noise into the feature states; and the per-step transform of all features costs more compute. This argument rests on reasoning rather than measurement, however: we did not evaluate a robocentric variant, and comparing the two anchorings, particularly on more dynamic trajectories, remains future work (Section VII).

F. Results in light of conditions

Although the proposed approach is generic, the flight conditions are narrow: speeds below 1 m/s, limited maneuvering, and a single planar feature wall. The planar wall keeps the feature depths similar, so a single depth prior fits them all. But it comes with other costs: it provides weak parallax and tightens trackable features to a smaller vertical feature spread (Figure 6). A planar scene can create ambiguity between yaw and lateral motion, since the two produce similar image flow; the same holds for pitch and vertical motion. Weak parallax also under-constrains translation, which is consistent with the observed position drift in the results. Depth is additionally shaped by the initialization prior (Section IV): under weak parallax the visual updates act almost entirely through the

feature bearings, while the depths are only weakly observed and stay near their tight initialization. The trajectory is in this sense decoupled from the feature depths: its shape follows the bearings and the IMU-driven prediction, and its metric scale is carried primarily by the predictor’s gravity term, with the depth initialization mattering only when set far from the true scene, rather than by parallax. Sweeping the fallback prior (Figure 34) and the initialization tightness (Figure 35) bounds this dependence from both sides: accuracy changes little across the plausible depths and over an order of magnitude of initialization tightness, degrades steeply when the prior places features far beyond the true scene, and breaks down once the initialization becomes nearly uninformative, suggesting that parallax alone cannot establish scale in this regime. How much scale could instead be recovered visually remains an open question, and would need a scene with stronger depth variation to test.

VII. FUTURE WORK

The directions below follow the findings of the discussion: the front-end as a impactful stage, the back-end estimation, and the flight conditions under which the pipeline was evaluated.

A. Front-End Improvements

As discussed in Section VI, the front-end is both the dominant computational cost and the significant accuracy influence of the current pipeline. The following directions may improve its robustness and consistency, some tie into unique properties of FWMVs.

Adaptive front-end parameters: A first direction is adaptive parameter tuning of the ARC corner detection stage. In the current implementation, the ARC test operates with fixed arc-length thresholds and a fixed refractory threshold κ selected by hand tuning on the platform. Because event generation is strongly influenced by wingbeat-induced motion, adapting these parameters according to the live on-board flapping frequency may improve corner detection consistency across different flight regimes. The analysis (Appendix A) showed that the relationship between the temporal filtering window and the dominant oscillation period $T_{\text{event_osc}}$ significantly affects the characteristics of detected corners; an adaptive setup using this relationship could for this reason be particularly beneficial. Such an approach would likely introduce only minimal additional computational cost.

Similarly, future work could investigate adaptive clustering strategies. The current spatiotemporal clustering parameters were tuned for the experimental platform and flight conditions considered in this study. In future work, both the temporal deactivation threshold r_t and the spatial radii (r_x, r_y) could be adjusted dynamically using information such as flapping frequency, estimated feature depth, or feature-state uncertainty. In particular, integrating cluster formation more tightly with the EKF back-end may allow clustering parameters to adapt to the predicted state of individual features.

Lightweight spatial feature priors: A second direction is the incorporation of spatial feature priors into the asynchronous event-feature tracking pipeline. The current approach associates events to feature tracks primarily through the spatiotemporal radius clustering and constant-velocity Kalman tracking stages, by design checking on local temporal and spatial consistency, but not encoding their specific structure. Probabilistic data-association methods, such as expectation-maximization-based feature tracking [41], may provide a mechanism for modeling the spatial structure of features over time while maintaining robustness against incorrect associations. A big caveat however is that such methods must account for the periodic visual motion introduced by flapping-flight, as features observed during different phases of the wingbeat cycle may show substantially different event patterns, as evidenced by the strongly oscillatory event structure observed in the analysis (Appendix A).

Line-based features: A further improvement is the inclusion of line-based features in addition to the compact spatiotemporal cluster detections used in this work. Often environments contain strong linear structures that remain observable even when corner features become sparse. Lightweight event-based line detection and tracking methods [42] could thus complement the existing front-end and improve feature availability in low-texture environments. Other previous work has already demonstrated the successful integration of point and line features within visual-inertial odometry systems [31].

Data-driven feature tracking: Another promising direction is the use of lightweight learning-based feature tracking. Recent progress in event-based neural processing has demonstrated robust feature tracking capabilities [39]. As embedded hardware continues to improve, small neural networks running on dedicated NPUs may provide more adaptive tracking while remaining compatible with the severe SWaP constraints of FWMV platforms.

Joint front-end and back-end tuning: The front-end parameters were hand-tuned on the vehicle, while the back-end noise parameters were optimized separately against the motion-capture reference (Appendix D-A, Appendix C). Optimizing both jointly against the same trajectory-error objective requires no hardware or algorithmic change, only changes to the raw event recording code. This would let the front-end trade-offs between feature yield and stability be resolved by estimation accuracy directly, likely leading to a more robust and accurate system.

B. Back-End State Estimation Improvements

Full on-board loop closure: The closed-loop hover injected the horizontal VIO position into the platform's stock firmware EKF, with height held on an external reference (Section V). The natural next step is closing the loop natively: tuning the controller to fly directly on the VIO estimate, and extending the loop to the vertical axis. This would remove the remaining external dependency and turn the demonstrated hover into fully self-contained flight supported by nothing but the presented VIO system. A lightweight time-of-flight range sensor could serve as an additional height input, adding robustness.

Incorporating higher-level event information: The proposed EKF system by design only relies on sparse feature observations from the front-end pipeline. An extension would be to incorporate spatial event information directly into the measurement update process. Similar in spirit to the spatial feature priors discussed for the front-end, event-alignment measurements, such as event volume consistency or local event-structure as update residuals, could provide additional constraints on the estimated state. This work stayed away from such approaches, as for reasons outlined in the analysis (Appendix A), but they may offer a promising direction for both flapping-wing and conventional MAV platforms when implemented accounting for phase and flapping frequency.

Adapted short-window motion estimation: The analysis showed that short-window alignment methods such as plane fitting and contrast maximization mainly recover the flapping-induced motion on this data (Appendix A). The most direct way to make them viable is inertial derotation: the on-board gyroscope measures the dominant rotation directly, and inertially motion-compensating the events before alignment is common practice in event-based VIO [32]. This could expose the weaker translational motion these methods currently fail to recover. The filter also estimates yaw worst (Section VI), so cancelling the rotation could help most on that axis. Since the gyroscope and events are already co-sampled on-board, this is directly feasible, though whether it survives the noise on raw gyro rates remains an open question. Contrast maximization may also be worth revisiting with flapping-aware adaptations, such as time-dependent flow parameterizations or phase-aware objectives; combined with derotation, it has the potential to be more robust to environmental differences than the corner-based front-end used here, and could complement rather than replace it.

C. More Challenging Flight Conditions

The evaluation presented in this work focused on controlled flight conditions with a limited range of maneuvers and environmental variability. The most direct extension is a scene with more depth variation: it would strengthen the translation constraint, reduce the pitch/vertical and yaw/lateral ambiguities identified in Section VI, and test a level of generality the current setup cannot. Lower contrast and less distinct features would further approximate the environments in which FWMVs might operate. Further directions include aggressive flight maneuvers with larger angular velocities and environments with greater feature variability. These conditions are expected to alter event generation statistics significantly and may expose limitations of the current parameterization. In particular, faster flight will intensify the oscillatory event patterns identified in the analysis (Appendix A), potentially stressing both the ARC corner detector and the spatiotemporal clustering stages. Continuously-translating trajectories are also the regime in which robocentric feature anchoring becomes attractive (Section VI), making it worth revisiting alongside more complex environments.

VIII. CONCLUSION

This work shows that event cameras can support on-board visual-inertial state estimation on a small, high-frequency flapping-wing platform, a regime in which the visual signal is dominated by wingbeat-induced oscillatory motion and the locally linear motion assumptions underlying existing event-based VIO methods break down. An analysis of event streams recorded on the platform showed that short-window alignment methods such as plane fitting and contrast maximization, in their vision-only form, recover mainly this high-frequency rotational motion rather than the platform's translation. The proposed pipeline for this reason avoids short-window alignment: an asynchronous front-end suppresses redundant events, filters arc-supported corner events, groups them into compact spatiotemporal clusters, and tracks them with a lightweight constant-velocity model. The resulting feature tracks are fused with inertial measurements in a tightly coupled Extended Kalman Filter using inverse-depth feature states and a low-order prediction model whose drag and coupling terms are adapted from existing flapping-wing dynamic models.

Evaluated on flapping-flights against a motion-capture reference, the front-end maintained feature tracks with a median lifetime of 4.68 s, spanning many wingbeats despite the oscillatory motion. The filter tracked world-frame velocity (0.23–0.31 m/s RMSE per axis) reasonably, with attitude estimated to within 7–9° for roll and pitch and less accurately in yaw (about 15°), while the absolute position accumulated drift (mean ATE of 0.75 m). The complete pipeline ran in real time on the on-board microcontroller at approximately 89 Hz with a sensing-and-compute payload of roughly 21 g, with the ARC corner detection stage, rather than the filter, constituting the dominant computational cost.

These results were obtained under controlled conditions, with low speeds, limited maneuvering, and a wall-facing planar feature setup. Within these conditions, a closed-loop flight test showed that the on-board horizontal-position estimate is accurate enough to stabilize a stationary hover (height held on an external reference), the first demonstration of an on-board event-based VIO estimate closing the control loop of a flapping-wing platform of this size. The identified directions for improvement follow directly from this evaluation: strengthening the front-end, which is a main computational bottleneck, closing the loop on the vertical axis as well, implementing flapping-aware feature association, and extending the evaluation to more aggressive maneuvers and more varied environments.

Overall the findings show that the same oscillation-dominated event stream that undermines common event-based estimation assumptions can still support lightweight state estimation when the front-end and back-end are designed for that motion regime. The results still leave room for improvement, but the work is a concrete first step: an on-board estimate already accurate enough to close a hover loop.

ACKNOWLEDGMENT

Research rarely follows a straight trajectory. Stein Stroobants, thank you for launching this one with great enthusiasm for

FWMAVs and a simple philosophy: just try it. Christophe De Wagter and Guido De Croon, thank you for the reality checks and drift corrections along the flight trajectory of this project. I am very grateful that I was allowed to work with such cool hardware and in such a fun work environment. Austin Lopez, thank you for the replacement-part pitstops along the way. Jens Kober, thank you for making sure I did not miss out on any crucial flight events. And, as with any journey, it is the people at home who give you the strength to keep on going: my family, my housemates, my closest friends, and my girlfriend.

REFERENCES

- [1] D. Floreano and R. J. Wood, "Science, technology and the future of small autonomous drones," *Nature*, vol. 521, no. 7553, 2015.
- [2] S. Rafee Nekoo, R. Rashad, C. De Wagter, S. B. Fuller, G. de Croon, S. Stramigioli, and A. Ollero, "A review on flapping-wing robots: Recent progress and challenges," *The International Journal of Robotics Research*, 2025.
- [3] G. C. H. E. De Croon, K. M. E. De Clercq, R. Ruijsink, B. Remes, and C. De Wagter, "Design, aerodynamics, and vision-based control of the DelFly," *International Journal of Micro Air Vehicles*, vol. 1, no. 2, pp. 71–97, 2009.
- [4] "Flapper nimble+ bioinspired flapping wing development platform," <https://flapper-drones.eu/nimbleplus/>, accessed: 2026-01-13.
- [5] R. Tapia, J. P. Rodríguez-Gómez, J. A. Sanchez-Diaz, F. J. Gañan, I. G. Rodríguez, J. Luna-Santamaria, J. R. Martínez-de Dios, and A. Ollero, "A comparison between frame-based and event-based cameras for flapping-wing robot perception," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.
- [6] A. Gómez Eguíluz, J. P. Rodríguez-Gómez, J. L. Paneque, P. Grau, J. R. Martínez-de Dios, and A. Ollero, "Towards flapping wing robot visual perception: Opportunities and challenges," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, 2019.
- [7] C. De Wagter, S. Tijmons, B. D. W. Remes, and G. C. H. E. de Croon, "Autonomous flight of a 20-gram flapping wing MAV with a 4-gram onboard stereo vision system," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 4982–4987.
- [8] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 db 15 μ s latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, 2008.
- [9] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, 2022.
- [10] R. Tapia, J. Luna-Santamaria, I. G. Rodriguez, J. P. Rodríguez-Gómez, J. R. Martínez-de Dios, and A. Ollero, "Flight of the future: An experimental analysis of event-based vision for online perception onboard flapping-wing robots," *Advanced Intelligent Systems*, 2025.
- [11] A. G. Eguíluz, J. Rodríguez-Gómez, R. Tapia, F. Maldonado, J. Acosta, J. Martínez-de Dios, and A. Ollero, "Why fly blind? event-based visual guidance for ornithopter robot flight," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 1958–1965.
- [12] J. P. Rodriguez-Gomez, R. Tapia, J. L. Paneque, P. Grau, A. Gomez Eguíluz, J. R. Martinez-de Dios, and A. Ollero, "The GRIFFIN perception dataset: Bridging the gap between flapping-wing flight and robotic perception," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1066–1073, 2021.
- [13] E. Mueggler, C. Bartolozzi, and D. Scaramuzza, "Fast event-based corner detection," in *Proceedings of the British Machine Vision Conference (BMVC)*, Sep 2017.
- [14] I. Alzugaray and M. Chli, "Asynchronous corner detection and tracking for event cameras in real time," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, Oct 2018.
- [15] M. Litzenberger, C. Posch, D. Bauer, A. N. Belbachir, P. Schön, B. Kohn, and H. Garn, "Embedded vision system for real-time object tracking using an asynchronous transient vision sensor," in *2006 IEEE 12th Digital Signal Processing Workshop & 4th IEEE Signal Processing Education Workshop*, 2006, pp. 173–178.

- [16] F. Barranco, C. Fermuller, and E. Ros, "Real-time clustering and multi-target tracking using event-based sensors," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 5764–5769.
- [17] Z. Wang, T. L. Molloy, P. van Goor, and R. Mahony, "Asynchronous blob tracker for event cameras," *IEEE Transactions on Robotics*, vol. 40, pp. 4750–4767, 2024.
- [18] D. Gehrig, H. Rebecq, G. Gallego, and D. Scaramuzza, *Asynchronous, Photometric Feature Tracking Using Events and Frames*, 2018. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-01258-8_46
- [19] I. Alzugaray and M. Chli, "HASTE: Multi-hypothesis asynchronous speeded-up tracking of events," in *31st British Machine Vision Conference (BMVC)*, 2020.
- [20] P. Li, H. Yao, Z. Dai, and X. Zhu, "Asynchronous visual-inertial odometry for event cameras," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLVIII-4-2024, 2024, source identifying LK and event frame usage.
- [21] J. P. Rodriguez-Gomez, A. Gomez Eguiluz, J. R. Martinez-de Dios, and A. Ollero, "Asynchronous event-based clustering and tracking for intrusion monitoring in UAS," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020.
- [22] R. Benosman, C. Clercq, X. Lagorce, S.-H. Ieng, and C. Bartolozzi, "Event-based visual flow," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 407–417, 2014.
- [23] M. T. Aung, R. Teo, and G. Orchard, "Event-based plane-fitting optical flow for dynamic vision sensors in fpga," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.
- [24] S. Shiba, Y. Klose, Y. Aoki, and G. Gallego, "Secrets of event-based optical flow, depth, and ego-motion by contrast maximization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.
- [25] J. Hagenaaers, Y. Wu, F. Paredes-Vallés, S. Stroobants, and G. de Croon, "On-device self-supervised learning of low-latency monocular depth from only events," in *arXiv preprint arXiv:2412.06359*, 2025. [Online]. Available: <https://arxiv.org/abs/2412.06359>
- [26] F. Paredes-Vallés, K. Y. W. Scheper, C. De Wagter, and G. C. H. E. de Croon, "Taming contrast maximization for learning sequential, low-latency, event-based optical flow," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023.
- [27] M. W. Mueller, M. Hamer, and R. D'Andrea, "Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadcopter state estimation," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1730–1736.
- [28] M. W. Mueller, M. Hehn, and R. D'Andrea, "Covariance correction step for kalman filtering with an attitude," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 5, pp. 1137–1148, 2016.
- [29] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 298–304.
- [30] B. Kueng, E. Mueggler, G. Gallego, and D. Scaramuzza, "Low-latency visual odometry using event-based feature tracks," 10 2016, pp. 16–23.
- [31] W. Guan, P. Chen, Y. Xie, and P. Lu, "PI-evio: Robust monocular event-based visual inertial odometry with point and line features," *IEEE Transactions on Automation Science and Engineering*, vol. 21, no. 4, pp. 6277–6293, 2024.
- [32] A. R. Vidal, H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Ultimate SLAM? combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 994–1001, 2018.
- [33] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based visual inertial odometry," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [34] E. Mueggler, G. Gallego, H. Rebecq, and D. Scaramuzza, "Continuous-time visual-inertial odometry for event cameras," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1425–1440, 2018.
- [35] K. M. Kajak, M. Karásek, Q. P. Chu, and G. C. H. E. de Croon, "A minimal longitudinal dynamic model of a tailless flapping wing robot for control design," *Bioinspiration & Biomimetics*, vol. 14, no. 4, p. 046008, 2019.
- [36] M. Karásek, F. T. Muijres, C. De Wagter, B. D. W. Remes, and G. C. H. E. de Croon, "A tailless aerial robotic flapper reveals that flies use torque coupling in rapid banked turns," *Science*, vol. 361, no. 6407, pp. 1089–1094, 2018.
- [37] D. Czech and G. Orchard, "Evaluating noise filtering for event-based asynchronous change detection image sensors," in *2016 6th IEEE International Conference on Biomedical Robotics and Biomechatronics (BioRob)*, 2016, pp. 19–24.
- [38] V. Padala, A. Basu, and G. Orchard, "A noise filtering algorithm for event-based asynchronous change detection image sensors on truennorth and its implementation on truennorth," *Frontiers in Neuroscience*, vol. 12, 2018. [Online]. Available: <https://www.frontiersin.org/journals/neuroscience/articles/10.3389/fnins.2018.00118>
- [39] N. Messikommer, C. Fang, M. Gehrig, G. Cioffi, and D. Scaramuzza, "Data-driven feature tracking for event cameras with and without frames," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 47, no. 5, 2025. [Online]. Available: <http://dx.doi.org/10.1109/TPAMI.2025.3536016>
- [40] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *Int. J. Rob. Res.*, vol. 36, no. 10, p. 1053–1072, Sep. 2017. [Online]. Available: <https://doi.org/10.1177/0278364917728574>
- [41] A. Z. Zhu, N. Atanasov, and K. Daniilidis, "Event-based feature tracking with probabilistic data association," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017.
- [42] L. Everding and J. Conradt, "Low-latency line tracking using event-based dynamic vision sensors," *Frontiers in Neurobotics*, 2018. [Online]. Available: <https://www.frontiersin.org/articles/10.3389/fnbot.2018.00004>
- [43] Prophesee, "Plane fitting flow algorithm," https://docs.prophesee.ai/stable/algorithms/optical_flow.html#planefittingflowalgorithm, prophesee Metavision SDK Documentation, accessed February 2026.
- [44] F. Hamann, Z. Wang, I. Asmanis, K. Chaney, G. Gallego, and K. Daniilidis, "Motion-prior contrast maximization for dense continuous-time motion estimation," in *European Conference on Computer Vision (ECCV)*, 2024.
- [45] G. Gallego, H. Rebecq, and D. Scaramuzza, "A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [46] F. Paredes-Vallés, J. J. Hagenaaers, J. Dupeyroux, S. Stroobants, Y. Xu, and G. C. H. E. de Croon, "Fully neuromorphic vision and control for autonomous drone flight," *Science Robotics*, vol. 9, no. 90, p. eadi0591, 2024.
- [47] S. A. Bahnam, C. De Wagter, and G. C. H. E. de Croon, "Improving the computational efficiency of rovio," *Unmanned Systems*, vol. 12, no. 03, pp. 589–598, 2024. [Online]. Available: <https://doi.org/10.1142/S2301385024410012>

APPENDIX A

ANALYSIS OF EVENT STREAM CHARACTERISTICS

This appendix analyzes how the platform's oscillatory motion affects the event data characteristics and common parts of the event-based motion estimation pipelines, including plane-fitting, feature detection, and learning-based methods. The goal is to identify which assumptions and methods are most affected by the dynamics of flapping-wing flight.

Data collection and setup

Data was collected using a Flapper Nimble+ platform [4] equipped with a GenX320 event-based camera and an OpenMV H7 Plus microcontroller. The GenX320 is a lightweight (~3 g) event camera optimized for low-power applications, shown in Figure 14. The H7 Plus (~17 g) logs the raw event stream and was used for this recording because its larger memory prevents recording gaps at high event rates; the proposed pipeline itself runs on the faster OpenMV N6 used in all other experiments (Section V), a board of comparable size and weight. Additionally, a small bracket (~1 g) mounts the parts to the platform, bringing the total added payload to 21 g. Due to the microcontroller's computational constraints, the GenX320 thresholds were configured to limit the event rate to 1 million events per second.

Data collection was conducted in the CyberZoo at the TU Delft under well-lit conditions. Flight maneuvers in the data

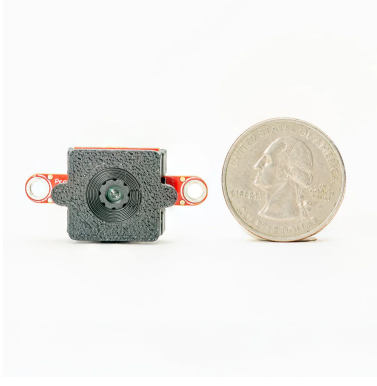


Fig. 14: GenX320 event-based camera shown next to a quarter for scale.

include hovering, forward/backward flight, lateral flight, and free flight, capturing a diverse range of motion profiles. To isolate the effect of flapping-induced dynamics, we additionally recorded handheld data in which the platform was moved manually along similar trajectories under identical camera settings, but without active wing flapping. This allows us to control for trajectory and scene content while removing flapping-induced oscillations. Throughout this appendix, we refer to these two conditions as the *flapping-flight* and *non-flapping* datasets.

Figure 15 shows the experimental setup mounted on the platform (total weight of 21 g) within the CyberZoo, as well as an overview of the CyberZoo test area where the data was recorded.

Event characteristics on FWMAV

a) *Faster flapping produces more events.*: Throughout the recordings we consistently observed higher event rates at higher flapping rates, although this relationship was not captured systematically enough to plot. The observation is in line with the hypothesis posed for a larger flapping-wing platform, namely that the oscillations generated during flapping-wing flight correlate with the number of generated events [6]. This indicates that more aggressive wing motion produces stronger brightness changes in the scene, leading to higher event activity.

b) *Dominant temporal structure.*: A more detailed analysis of the spatiotemporal structure of the event data reveals pronounced oscillatory patterns generated by the flapping-wing dynamics. Figure 16 illustrates the evolution of events in the (x, y, t) volume over 100 ms, where the FWMAV hovered in front of the TU Delft logo in the CyberZoo. Clear oscillations in both x and y are visible in the event distribution over time, consistent with periodic body motion induced by the flapping wings.

To further analyze the temporal structure, Figure 17 shows the event count over time for both polarities. A clear oscillatory pattern is visible in the event rate; its periodicity, near twice the flapping frequency (analyzed below), indicates that a flapping-induced oscillation dominates the camera motion. In the motion-capture reference (Section V, Figures 7 and 8) this oscillation is most visible in the yaw axis, although the flapping also induces periodic motion on the other axes.

Let f_{flap} denote the fundamental flapping frequency of the platform and $T_{\text{flap}} = 1/f_{\text{flap}}$ the corresponding flapping period. For the platform used in this work, f_{flap} lies approximately between 12–17 Hz.

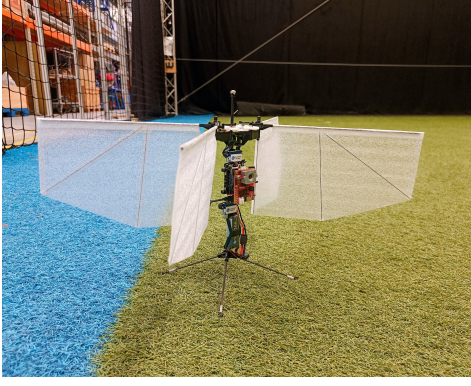
Although the wings flap at this fundamental frequency, the oscillatory structure visible in the event-rate signal contains higher-frequency components than one might initially expect from the flapping frequency alone. Two effects can explain this, each producing a component near twice the flapping frequency on its own. First, for body yaw oscillating approximately sinusoidally at the wingbeat frequency, the angular velocity reaches its maximum magnitude twice per cycle; since event generation scales with the magnitude of the image velocity, the event rate peaks twice per oscillation cycle and thus oscillates at twice that frequency. Second, the aerodynamic force scales with the square of the wing velocity, which for sinusoidal wing motion forces the body directly at twice the flapping frequency; a simplified derivation is given in Appendix B, and the camera then observes this doubled body oscillation.

Under this model, the dominant image oscillation appears at $f_{\text{event_osc}} = 2 f_{\text{flap}}$, matching the periodicity visible in Figure 17 and in the mocap-derived yaw rate. Because an event camera responds most strongly to fast image motion, the event rate does not peak at the turning points of the oscillation, where the image is momentarily stationary, but in the middle of each left–right sweep, where the image velocity is highest. These velocity maxima occur twice per oscillation cycle, so the event-rate spikes recur at $f_{\text{spike}} = 2 f_{\text{event_osc}} = 4 f_{\text{flap}}$. We emphasize that this is a simplification of the flapping dynamics rather than a confirmed description of the event data: the derivation assumes symmetric, purely sinusoidal wing motion, whereas on the real platform the left and right wing pairs do not move identically, so the recorded event stream also contains other harmonic components. The predicted sinusoidal pattern is nevertheless clearly visible in the data of Figure 16 and Figure 17. This simplified structure is what we later use when verifying several front-end parameters.

c) *High frequency oscillations dominate the event stream.*: Consequently, even though the wings flap at f_{flap} , the resulting event stream exhibits a pronounced oscillatory structure with significant higher frequency content. This has important implications for event-based perception methods, which often rely on assumptions of locally consistent motion within temporal windows. For this reason we evaluate the methods over a broad range of temporal windows to understand how the interaction between flapping dynamics and method assumptions affects performance. We refer to the frequency of the dominant oscillatory component in the event stream as $f_{\text{event_osc}}$, and to its period as $T_{\text{event_osc}}$.

How do FWMAV dynamics affect event-stream characteristics?

Event rate modulation. The event rate increases with flapping rate, consistent with expectations from prior FWMAV work. This implies that more aggressive motion can increase the available event signal, suggesting a coupling between actuation dynamics and perceptual

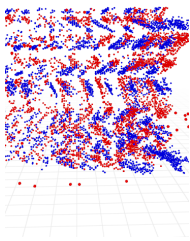


(a) Flapper Nimble+ platform, GenX320 event camera, OpenMV H7 Plus microcontroller mounted

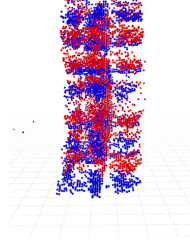


(b) Overview of the CyberZoo test area at the TU Delft, featuring the TU Delft logo.

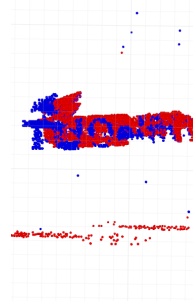
Fig. 15: Experimental setup and testing environment.



(a) Events over 100 ms viewed along the y axis.



(b) Events over 100 ms viewed along the x axis.



(c) Events over 100 ms viewed along the time axis.



(d) Events accumulated over 10 ms (frame view).

Fig. 16: Event visualizations through time from different viewpoints in (x, y, t) over 100 ms, and a 10 ms accumulation frame.

richness.

Frequency amplification effects. The dominant oscillation frequency can exceed the fundamental wingbeat frequency. This is plausibly explained by the aerodynamic force profile over a wingbeat, which introduces strong components near twice the flapping frequency, resulting in higher-frequency components in the observed event stream.

Non-linear spatiotemporal structure. The event stream exhibits a complex spatiotemporal structure dominated by high-frequency oscillations. Even in very short time windows, it appears much more non-linear than the non-flapping dataset.

Plane-fitting

Plane fitting estimates local normal flow by fitting planes to events in the (x, y, t) volume under a short-time locally constant motion assumption. For the analysis below, we use a local plane-fitting pipeline to compute normal flow on the FWMAV data. We additionally observed that the Plane Fitting Flow implementation in the Prophesee Metavision SDK [43] shows the same dominant directional structure and similar coherence trends on these sequences.

When visually inspecting the normal flow computed across both datasets, a clear difference can be observed. Plane fitting applied to flapping-flight event data produces normal flow

fields that appear more spatially inconsistent (Figure 18a), with directions that vary rapidly over time. This behavior is observed across a broad range of temporal plane-fitting windows T . In contrast, the normal flow estimated from the non-flapping dataset appears substantially more consistent (Figure 18b), with vectors predominantly pointing in a common direction over time.

One expects alternating flow directions due to the periodic body rotations induced by flapping. However, the flow estimated from the flapping-flight data does not only exhibit structured oscillation; it appears spatially fragmented and poorly aligned within individual temporal windows as well.

To quantify this observation, Figure 18c shows the spatial coherence of the estimated normal flow, computed over short time bins across a 10×10 grid in the camera frame. Spatial coherence measures the directional alignment of flow vectors within each bin (0 = random, 1 = perfectly aligned). Across all temporal plane-fitting windows T , the flapping-flight data shows consistently lower spatial coherence than the non-flapping dataset, indicating reduced directional alignment and frequent directional changes in the estimated flow field.

We can examine this behavior by distinguishing three regimes relative to the dominant flapping period $T_{\text{event_osc}}$.

d) $T < T_{\text{event_osc}}$: For short temporal windows, coherence increases. Within very small windows, the rotational motion on the event data induced by flapping dynamics approaches

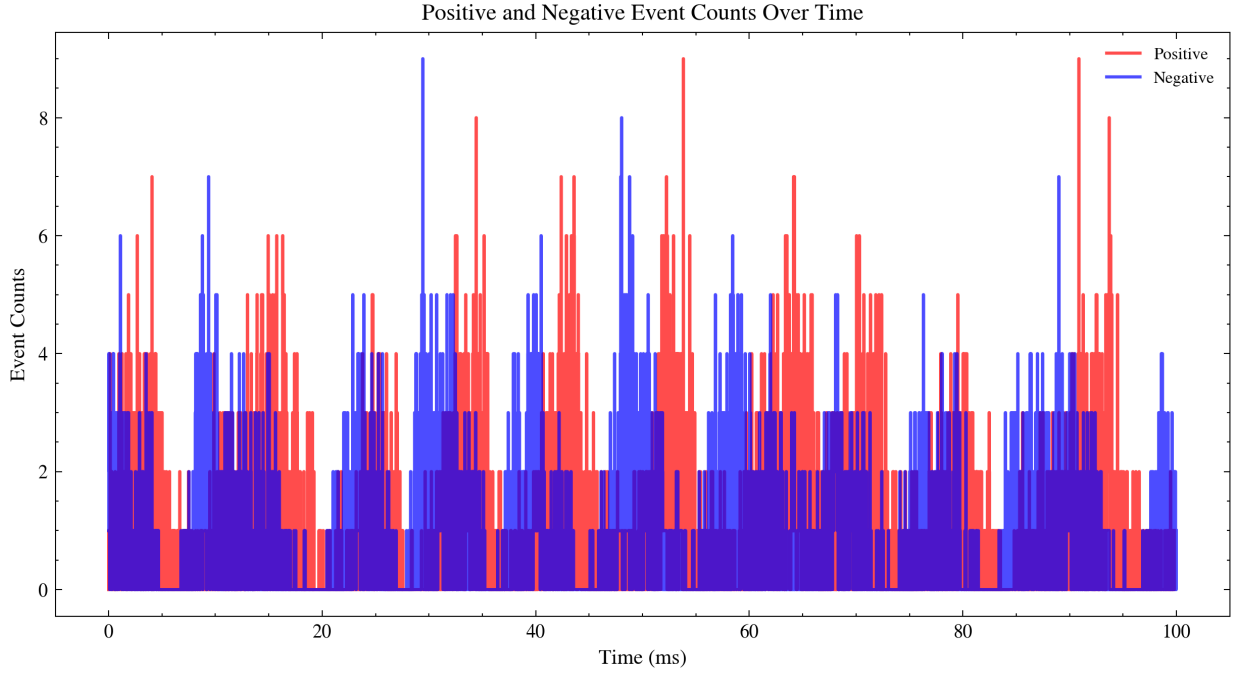
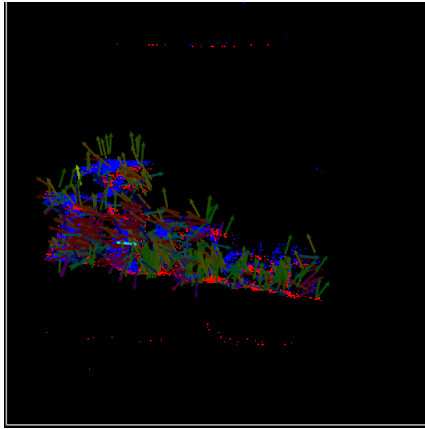
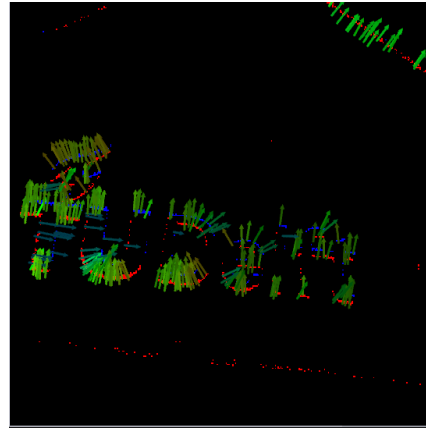


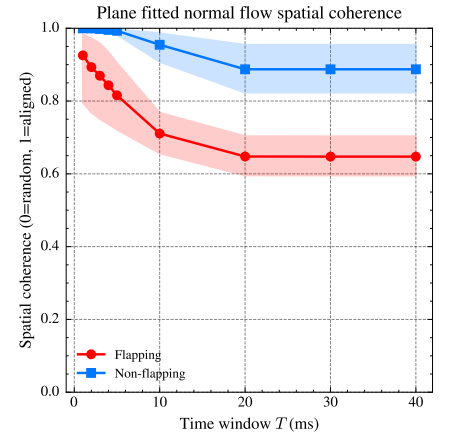
Fig. 17: Event count over time, separated by polarity (positive/ON events in red, negative/OFF events in blue). Events are accumulated in temporal bins of 2 ms.



(a) Estimated flow: flapping



(b) Estimated flow: non-flapping



(c) Spatial coherence vs. window T

Fig. 18: (Left) Example estimated flow field during flapping flight. (Middle) Example estimated flow field during non-flapping motion. (Right) Spatial coherence vs. temporal window T .

more linear behavior. The angular velocity changes less over such short intervals, resulting in a more constant motion field. As a result, the estimated flow vectors are more directionally aligned, leading to higher spatial coherence.

e) $T \approx T_{event_osc}$: As the temporal window increases and approaches the T_{event_osc} period in the event data, coherence decreases. Each window begins to include both positive and negative phases of the oscillatory yaw motion. Plane fitting attempts to estimate a single spatio-temporal plane across motion that reverses direction within the same window, implicitly assuming local motion consistency. This causes opposing flow directions to be mixed in the estimation, reducing directional alignment and thus lowering spatial coherence.

f) $T > T_{event_osc}$: For temporal windows larger than the T_{event_osc} period, coherence reaches a plateau. Once the window contains multiple complete oscillation cycles, increasing T does not introduce new directional structure; it merely adds repetitions of similar periodic motion. The estimated flow remains spatially inconsistent and coherence does not improve.

Taken together, plane fitting on flapping-flight data follows a clear short-window/long-window pattern. For short temporal windows, the locally linear approximation holds better and the estimated normal flow becomes more coherent, but it primarily captures the dominant high-frequency oscillatory rotation rather than weaker translational motion. Moreover, short time windows contain fewer events and hence increase

estimation uncertainty. As the temporal window approaches and exceeds $T_{\text{event_osc}}$, opposing motion phases are mixed within a single estimate, spatial coherence drops, and the method provides little additional useful motion information. In contrast, the non-flapping data yields spatially consistent flow fields aligned with the dominant direction of motion. These findings suggest that standard plane-fitting methods are intrinsically challenged by strongly periodic, high-frequency body dynamics, not because they fail immediately under flapping, but because the dominant oscillatory motion violates the locally consistent motion assumption once the temporal window becomes too large.

Learning-based with contrast maximization

Learning-based event motion estimation often uses contrast maximization, where a predicted motion field is used to warp events so that consistent motion produces a sharper accumulation. In this appendix, we evaluate that idea with TinyCmax [25], using the dataset-specific adaptations described in Appendix K. We test both the base and iterative-warping variants across temporal windows on the flapping-flight and non-flapping datasets to examine how well this formulation handles oscillatory flight. Unless stated otherwise, the models are trained from scratch; we additionally evaluated initialization from publicly available pretrained weights to assess transferability to the FWMAV setting.

Figure 19 shows the average validation loss of multiple runs over epochs for three different temporal windows. On non-flapping data, the loss decreases consistently across training for the 5 and 10 ms windows, indicating that the network successfully learns motion under smooth camera dynamics.

On flapping-flight data, in contrast, a modest but consistent decrease in validation loss is observed only for very short windows below ~ 3 ms (the 1 ms window in the figure), while for larger windows training quickly plateaus and shows little to no improvement over epochs.

We can examine this behaviour by considering the interaction between the oscillatory flapping dynamics and contrast maximization. For larger windows, events from multiple phases of the oscillatory wingbeat are accumulated within a single optimization interval. As the dominant rotational motion reverses direction within such windows, the assumption of locally consistent motion is violated. Consequently, the network cannot coherently align events using a single flow field, and the contrast objective provides a weak or inconsistent gradient signal.

For shorter windows (1 ms), the motion induced by flapping is locally more linear. Within such short intervals, angular velocity changes are limited, allowing the network to partially align events and achieve a measurable reduction in loss. Nevertheless, total loss reduction remains limited, plausibly for the same reason as in plane fitting: such short windows contain few events and hence little evidence for motion. This invites further investigation into what type of motion is being learned within these short windows, and whether the dominant oscillatory component is being captured at the expense of weaker translational cues.

Inspection of the learned flow fields (Figure 20) with a short temporal window setting confirms that the model primarily captures the high-frequency left–right oscillatory motion induced by flapping. This is visible in the figure as the dominant flow color flipping between the leftward and rightward panels, opposite directions on the color wheel; the patches of the opposite color within a single panel appear where the 10 ms window overlaps a reversal of the sweep. Translational components, which evolve at lower frequency and contribute less strongly to the contrast objective, are not recovered.

This mirrors the behavior observed for plane fitting. The TinyCmax model does not inherently fail to learn motion on flapping flight data. It converges to the dominant motion component that maximizes the contrast objective. In the FWMAV setting, this dominant component corresponds to periodic rotational oscillations, which generate strong event alignment signals. Lower-frequency translational motion contributes less to the loss and is therefore insufficiently recovered.

With the current maximization objective and under strongly periodic dynamics such as those observed in the flapper-flight data, contrast maximization appears to prioritize fast oscillatory components of motion while under-representing slower translational dynamics. Future work for methods using contrast maximization may benefit from: Incorporating motion priors, frequency aware objectives, or time-dependent flow models [24], [44] may help balance this effect and improve recovery of translational motion in oscillatory flight regimes.

Corner Detection

Event-based corner detectors identify spatiotemporal structures that can later serve as sparse motion cues in the form of trackable features. Here, we use eFast [13], an efficient detector that marks an event as a corner when the most recent events on a small pixel circle around it form a contiguous angular segment (an “arc”) of bounded length on the local time surface, as a representative low-latency feature detection front-end. The detector was applied to both the flapping-flight and non-flapping datasets to evaluate how oscillatory motion affects corner yield under arc-based spatial continuity constraints within a temporal window.

To quantify the effect of temporal accumulation, the corner yield was evaluated as a function of the temporal window T , defined as

$$\rho = \frac{N_{\text{corners}}}{N_{\text{events}}}.$$

Figure 21 shows the resulting corner yield for arc lengths of 8 and 10 pixels. Two regimes can be observed across both arc lengths:

- For short temporal windows, the non-flapping dataset yields slightly higher corner densities than the flapping-flight dataset.
- As the temporal window increases, the corner yield in the flapping-flight dataset increases and eventually equals and starts to exceed that of the non-flapping data.

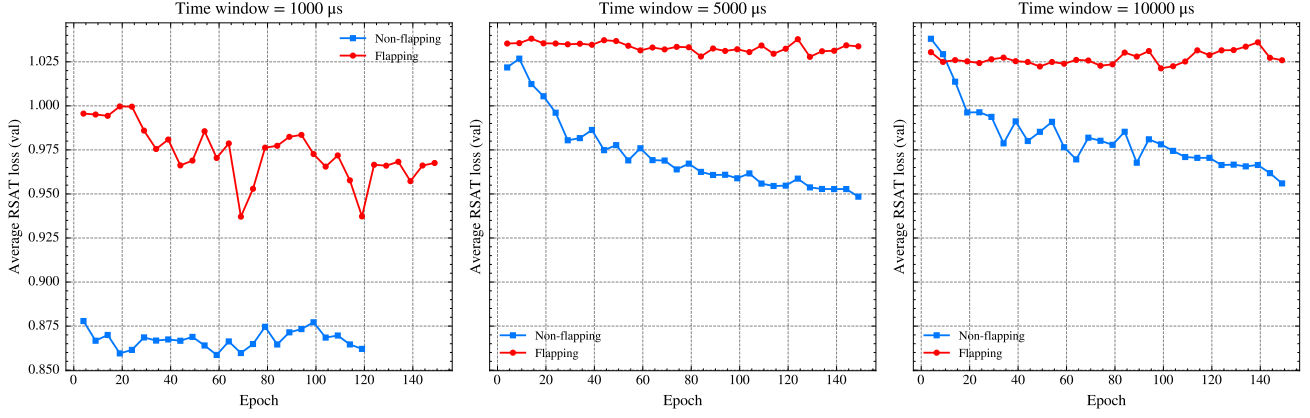


Fig. 19: Validation loss over epochs per temporal window ($\{1, 5, 10\}$ ms) for both flapping and non-flapping datasets.

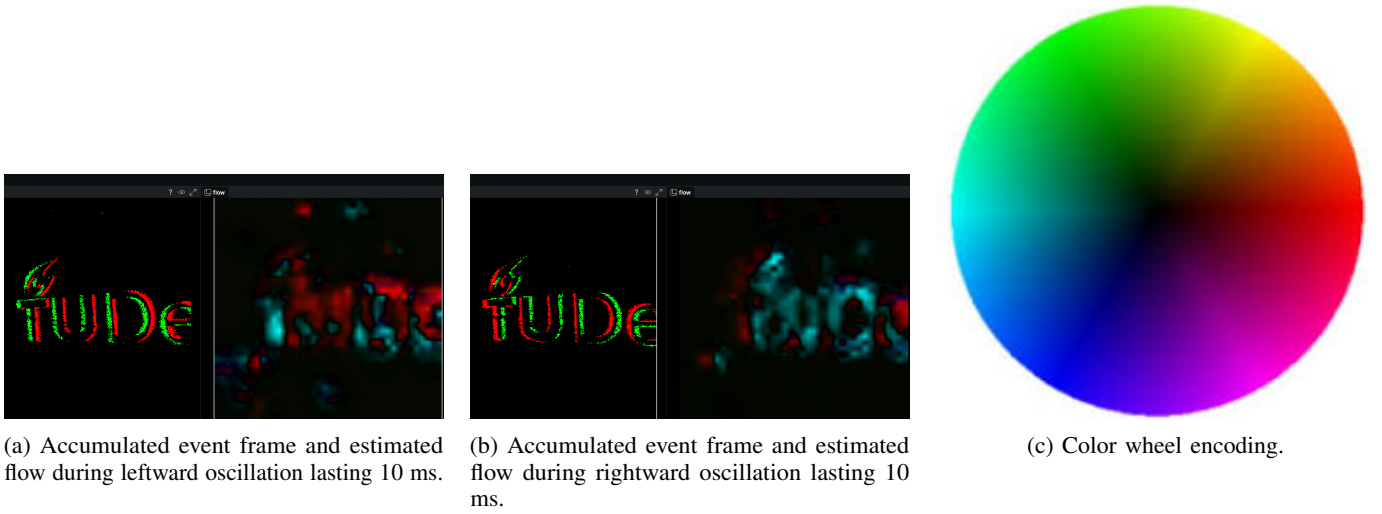


Fig. 20: Estimated flow fields from TinyCmax on FWMAV flapping flight data over 10 ms while hovering in front of the TU Delft logo.

For small T , high frequency non-linear motion in the flapping-flight event data can disrupt local spatiotemporal consistency. Since eFast relies on arc-based spatial continuity within a temporal window, this leads to a minor reduction in corner yield compared to the non-flapping case. However, the difference is relatively small, suggesting that corners can still be detected effectively during flapping flight.

As T increases and begins to exceed the dominant oscillation period $T_{\text{event_osc}}$, the temporal window spans multiple oscillation cycles of the flapping-flight dynamics. Events generated during opposing motion phases then accumulate within the same window, increasing local event density and spatial overlap. This temporal integration partially restores arc-consistency conditions, leading to increased corner yield in the flapping-flight dataset.

As expected, increasing the arc length from 8 to 10 pixels reduces the overall corner yield in both datasets, as stricter arc-continuity constraints must be satisfied.

Figure 22 shows eFast detections over a time window of 50 ms and arc length 8 pixels. Both time snippets are pointed at the same TU Delft logo under similar lighting and from a

similar viewing point. During flapping flight, features appear sparser, reflecting the non-linear motion that disrupts arc continuity, but they are also more dispersed across the x, y coordinates of the frame. In contrast, the non-flapping data produces denser features that are more spatially clustered.

Taken together, these results show that the effect of flapping-flight dynamics on arc-based corner detection is strongly time-window dependent. When the temporal window is shorter than the dominant oscillation period, rapid rotational motion changes the temporal ordering of events around a candidate corner and disrupts arc formation, reducing corner yield. When the window spans multiple oscillation cycles, temporal integration stabilizes the local event structure and improves corner yield. For FWMAV platforms and arc-based detectors, this makes the temporal window a critical design parameter: shorter windows may yield fewer but more dispersed features, whereas windows exceeding $T_{\text{event_osc}}$ may yield more features but with increased spatial clustering. The stability and trackability of these features over time remains an open question, motivating adaptive window selection for arc-based detectors informed by flapping frequency and

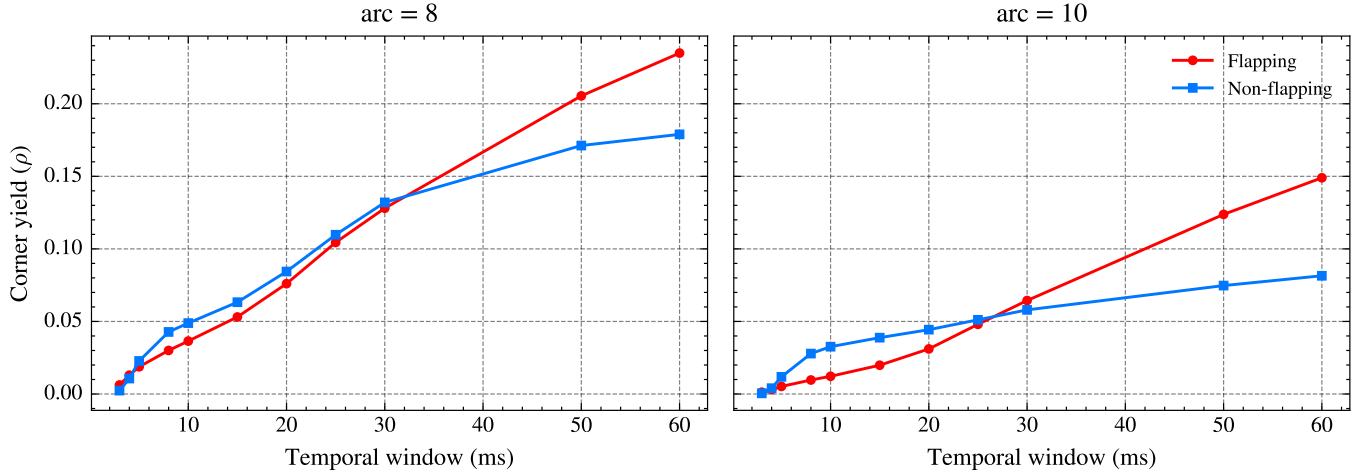
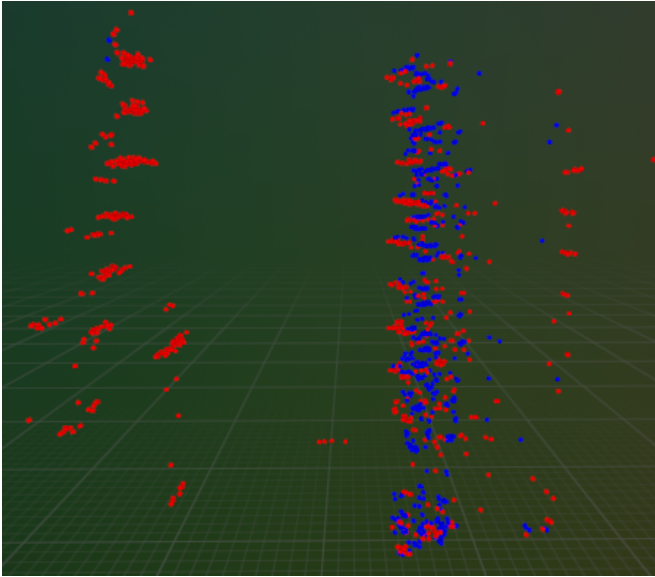
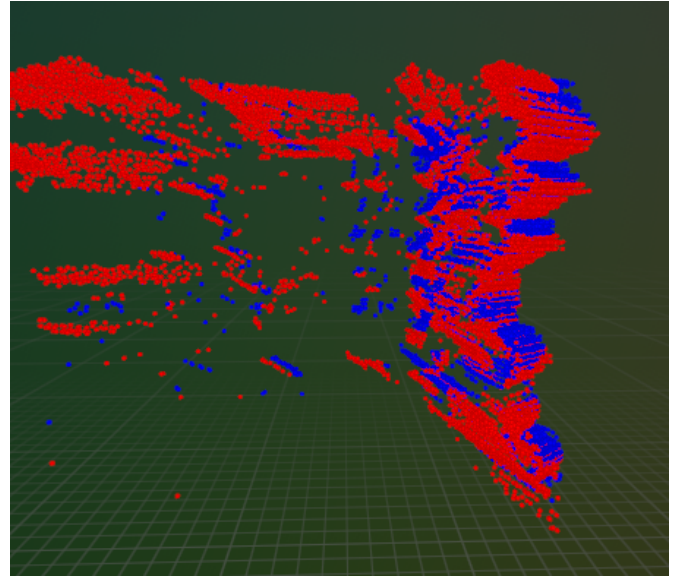


Fig. 21: Corner yield ρ as a function of temporal window T for flapping and non-flapping data, shown for arc lengths of 8 and 10 pixels.



(a) eFast features detected during flapping flight (50 ms window, arc of 8).



(b) eFast features detected during non-flapping motion (50 ms window, arc of 8).

Fig. 22: Comparison of eFast corner detection on flapping versus non-flapping FWMAV data. Every plotted point is a detected corner event, colored by polarity (positive in red, negative in blue). Flapping flight produces fewer corners, while non-flapping motion yields denser and more spatially clustered features.

feature tracking strategies robust to oscillatory dynamics for FWMAVs. In particular, because the corners are sparse, dispersed, and short-lived, aggregating them over space and time rather than tracking each corner individually is a natural way to recover stable feature measurements; this is the spatiotemporal clustering strategy adopted by the proposed front-end (Section IV).

Summary of the method evaluation

How do existing methods perform on event data recorded from FWMAVs?

Plane-fitting. For short temporal windows ($T < T_{\text{event_osc}}$), plane-fitting better satisfies the locally consistent motion assumption and can recover the dominant high-frequency oscillatory rotation induced by flapping. However, the recovered flow contains limited useful lower-frequency translational information for visual odometry. For longer windows ($T \geq T_{\text{event_osc}}$), opposing motion phases are mixed within a single estimate, spatial coherence drops, and the method provides little useful translational information.

Learning-based methods. Similarly, for short temporal windows, learning-based contrast maximization converges mainly to the dominant oscillatory motion component induced by flapping-flight. For larger windows, optimization stagnates because accumulated events violate the locally consistent motion assumption and weaken the contrast objective. As with plane-fitting, useful lower-frequency translational motion is not reliably recovered without adaptation of the objective.

Feature detection. For short windows, rapid high-frequency non-linear motion disrupts arc continuity, leading to slightly reduced corner yield. However, the remaining corners may still be useful for visual odometry if they prove stable and trackable over longer periods of time. For longer windows, corner density increases due to temporal integration across oscillation cycles, but the resulting features may be less distinctive. The stability and trackability of arc-filtered features over time remains an open question.

Which methods and assumptions fail under FWMAV dynamics? Both plane-fitting and learning-based contrast maximization assume that motion is locally consistent within a short temporal window. Under flapping-wing dynamics, this assumption is only approximately valid in very short windows, where both methods can partially recover the dominant oscillatory component.

For longer windows, the presence of multiple motion phases within a single window breaks this assumption. For plane-fitting, opposing directions are mixed within one spatio-temporal plane estimate. For contrast maximization, a single flow field can no longer coherently align the accumulated events.

For feature-based methods, the arc-based spatial continuity assumption is challenged by the non-linear flapping-flight motion. The observed corner-yield trends show sensitivity to time-window length, but whether the detected features remain sufficiently stable and trackable for visual odometry is still unresolved.

Which methods or adaptations appear most promising for robust motion estimation on FWMAVs?

Current results suggest that unadapted plane-fitting and learning-based contrast maximization methods mainly recover the dominant oscillatory motion and provide limited useful lower-frequency translational information for visual odometry. The most promising adaptations are therefore those that model the oscillation explicitly rather than suppress it. The $2f_{\text{flap}}/4f_{\text{flap}}$ structure used throughout this appendix is itself a simplified, first-order model; a more detailed harmonic analysis of the flapping-induced event structure, could directly inform the sinusoidal and phase-aware motion models suggested below.

For plane-fitting, replacing the locally linear plane assumption with sinusoidal and/or phase-aware motion models may be an interesting direction. For learning-based

approaches, incorporating time-dependent learnable flow parameterizations, sinusoidal priors, or phase-aware loss formulations may reduce bias toward dominant oscillatory components and enable recovery of translational motion. Feature-based methods may be comparatively promising in the short term because they still produce candidate features under flapping-flight conditions, but their practical value for motion estimation depends on whether those features remain stable and trackable over time. Adaptive window selection informed by flapping frequency, together with tracking strategies robust to oscillatory dynamics, may help assess and exploit that potential for motion estimation on FWMAVs. Aggregating the sparse corner detections into spatiotemporal clusters is one such strategy, trading the fine event structure for more stable, longer-lived feature measurements.

Computational constraints

The analysis above considered which motion assumptions break down under flapping flight. A second constraint comes from the platform itself: the camera, microcontroller, and bracket together form a payload of only 21 g in both the recording (H7 Plus, above) and flight (N6, Section V) configurations, and the on-board processing is performed by a microcontroller rather than a desktop CPU or GPU. To place existing event-based methods in this context, Table VI summarizes the compute hardware on which representative methods were demonstrated, as reported in the respective papers.

The front-ends and odometry systems are almost exclusively demonstrated on desktop or laptop CPUs and workstation GPUs, frequently offline; several papers report neither power nor hardware mass, because on-board deployment was not their objective. The systems that were actually flown carry an on-board computer of the mini-PC, embedded-GPU, or smartphone-class ARM category, on airframes of roughly 0.8–3.5 kg, an order of magnitude or more heavier than the platform considered here. On the frame-based side, filter-based VIO has been made efficient enough to run on a Raspberry Pi Zero 2W [47], underlining the suitability of the filter back-end for embedded use; it is the event-processing front-end that prior work has not brought to that scale.

This means that few of the existing event-based perception methods are currently deployable on our 21 g-payload system. The methods that report the lowest cost are the arc-based corner detectors [13], [14], which is the family the proposed front-end builds on (Section IV). Looking further ahead, neuromorphic processing is the most promising route to closing the gap: a fully neuromorphic event-vision-to-control pipeline has been flown on Intel’s Loihi processor at under one watt of processing power [46], a power budget compatible with a flapping platform. The remaining barrier there is integration mass rather than energy, since that demonstration still ran on a ~ 1 kg quadrotor. Together with the motion findings above, this compute budget motivates the lightweight, corner-based front-end and compact filter developed in this work.

TABLE VI: Reported compute hardware of representative event-based perception and odometry methods, as stated in their respective papers, placed against the platform used in this work.

Approach	Reported compute platform	Type	Flown on-board?
Arc-based corner detection [13], [14]	Intel i7 / Xeon laptop CPU, single-thread	CPU	No
Contrast-maximization optical flow [24], [26], [45]	Desktop CPU / workstation GPU, offline	CPU/GPU	No
Plane-fitting optical flow [23]	Xilinx Spartan-6 LX150 FPGA, 100 MHz	FPGA	No
Asynchronous feature tracking [18], [41]	Intel i7 desktop CPU, single-thread, offline	CPU	No
Data-driven feature tracking [39]	Workstation GPU (NVIDIA RTX 8000)	GPU	No
Asynchronous event clustering & tracking [21]	Intel NUC6i7 mini-PC, on-board	x86 mini-PC	Yes (~3.49 kg hexa.)
Filter-based VIO [29]	Intel i7 laptop CPU, single-core	CPU	No
Hybrid frame-event VIO [32]	Odroid XU4 (quad-core ARM), on-board	embedded ARM	Yes (quadrotor)
Point/line event VIO [31]	Intel NUC10i7 mini-PC, on-board	x86 mini-PC	Yes (~1.36 kg quad.)
On-device event depth [25]	NVIDIA Jetson Orin NX, on-board	embedded GPU	Yes (~0.82 kg quad.)
Fully neuromorphic vision & control [46]	Intel Loihi processor, on-board	neuromorphic	Yes (~1.0 kg quad.)
This work (Section IV)	OpenMV N6 microcontroller (Cortex-M55 @ 800 MHz)	MCU	Yes (21 g payload)

APPENDIX B

ORIGIN OF THE OSCILLATION COMPONENT

This appendix provides a simplified argument showing why flapping-wing dynamics can introduce a dominant component at twice the wingbeat frequency.

Assume quasi-steady aerodynamics and approximately sinusoidal wing stroke,

$$\theta(t) = \theta_0 \sin(\omega t), \quad \text{with } \omega = 2\pi f_{\text{flap}}.$$

The angular velocity becomes

$$\dot{\theta}(t) = \theta_0 \omega \cos(\omega t),$$

and the span-averaged relative wing velocity at equivalent radius R is

$$v_{\text{rel}}(t) = R\dot{\theta}(t) = R\theta_0 \omega \cos(\omega t).$$

Under quasi-steady assumptions, aerodynamic force scales with the square of the relative velocity:

$$F_{\text{aero}}(t) \propto v_{\text{rel}}^2(t) \propto \cos^2(\omega t).$$

Using

$$\cos^2 x = \frac{1}{2} (1 + \cos(2x)),$$

the force can be written as

$$F_{\text{aero}}(t) = \frac{K}{2} + \frac{K}{2} \cos(2\omega t),$$

where K groups constant terms.

The aerodynamic force therefore contains an oscillatory component at frequency

$$f_{\text{osc}} = 2f_{\text{flap}}.$$

This simplified model (sinusoidal stroke and a force scaling exactly with v_{rel}^2) yields a single oscillatory component at exactly $2f_{\text{flap}}$. Real wing kinematics and aerodynamics depart from these assumptions: the stroke is not perfectly sinusoidal, the force contains terms beyond the quasi-steady v^2 scaling, and the up- and down-strokes are likely not perfectly symmetric. The $2f_{\text{flap}}$ component nonetheless is dominant. This is consistent with the dominant event-stream oscillation $f_{\text{event_osc}} \approx 2f_{\text{flap}}$ identified in the analysis.

APPENDIX C

EKF PARAMETER TUNING

The EKF parameters were tuned by global black-box optimization against the motion-capture reference, minimizing a weighted trajectory error (position, orientation, and velocity contributions weighted 1.0, 0.3, and 0.5, with a 1 s settling window discarded at the start of each flight). The search used differential evolution (population 48, up to 10,000 evaluations, seed 42) over the 13 tuning flights, covering the process-noise spectral densities, the feature measurement noise, the admission and gating parameters (minimum updates, lost-frame timeout, innovation gate, gate warm-up), the initial inverse-depth uncertainty, and the initial state covariances. Continuous parameters were searched over one to two orders of magnitude, on a log scale where appropriate (e.g., accelerometer noise density 10^{-2} – 2 , feature measurement variance 10^{-3} – 10^{-1} , initial relative inverse-depth std 5×10^{-4} – 2×10^{-2}), the gate and warm-up linearly (8–40 and 0.1–9 s), and the integer admission parameters over 5–200 (confirmation) and 20–260 (timeout). The fallback inverse depth was held fixed at 0.28 (Section V), and the attitude-reversion term was left at its base value. The resulting values (Table VII) were applied unchanged to all reported datasets.

APPENDIX D

FRONT-END PARAMETER TUNING AND ABLATION STUDIES

This appendix lists the front-end operating values and reports the front-end ablations summarized in Section V-E. Each study varies one design choice or parameter while holding the rest of the pipeline fixed. Three findings summarize the outcome. First, every hand-set value behaves according to the mechanism it is meant to control, and no value sits at a poor operating point: two (the association gate and the track-death timeout) already sit at principled optima. Second, several parameters are conservative, with measurable head-room toward fewer, longer-lived features or lower compute. Third, and most consequential, the governing scales of most parameters are properties of the platform and scene, not of the algorithm: the flapping frequency sets the temporal windows, and the scene depth and feature size set the

TABLE VII: EKF and feature-management parameters used for all reported datasets. Process-noise entries are continuous-time spectral densities (Appendix G-D); ρ initialization uses the adaptive median depth of active features. The initial inverse-depth std is relative (dimensionless): the listed value is σ_ρ/ρ at initialization, so a feature at $\rho \approx 0.28$ enters with $\sigma_\rho \approx 2.7 \times 10^{-4} \text{ m}^{-1}$.

Parameter	Symbol	Value
<i>Process noise (spectral densities)</i>		
Accelerometer XY	$\sigma_{a,xy}$	0.253
Accelerometer Z	$\sigma_{a,z}$	0.0378
Velocity	σ_v	5.12×10^{-3}
Position	σ_p	3.74×10^{-3}
Attitude	σ_{att}	1.68×10^{-5}
Gyro roll/pitch	$\sigma_{g,rp}$	0.0682
Gyro yaw	$\sigma_{g,yaw}$	0.169
Attitude reversion	—	3.58×10^{-5}
<i>Measurement / feature initialization</i>		
Feature meas. variance	$\mathbf{R}$	5.45×10^{-3}
Initial rel. inverse-depth std	—	9.53×10^{-4}
Max. depth-uncertainty ratio	—	0.673
Fallback inverse depth	—	0.28
<i>Innovation gate / track management</i>		
Mahalanobis gate	γ	9.37
Gate warm-up	—	6.05 s
Min. updates to confirm	n_{birth}	96
Lost-frame timeout	—	137
<i>Initial state covariance (std)</i>		
Position XY	—	0.022 m
Position Z	—	0.014 m
Velocity	—	0.059 m/s

spatial ones, which motivates flap- and depth-aware adaptation as future work.

A. Front-end operating point

The front-end values were originally hand-tuned on the vehicle: the front-end was run on-board and its parameters adjusted visually for feature persistence during short evaluation flights. Rather than searching for globally optimal values, we selected operating points that maintained corner yield and track stability under the oscillatory motion while respecting the event-rate and compute limits of the on-board OpenMV implementation. A systematic tuning was not possible at the time, because logging the raw events and IMU simultaneously at a rate representative of closed-loop inference exceeds the on-board compute budget. The dedicated raw-event recordings used here close that gap and enable a one-at-a-time sensitivity analysis; they do not constitute a joint optimization (Section D-C).

In the main methodology, we report only the final operating values and short design rationales. The complete set of front-end operating values is listed in Table VIII; the GenX320 bias settings are reported separately in Appendix I.

B. Protocol and metrics

Each ablation varies one parameter and leaves all others at the on-board operating point (Table VIII; the tracker noise ratio is $q/r = 1.0/4.0 = 0.25$ and the processing cycle period

TABLE VIII: Front-end parameters used for all reported datasets. Symbols match the methodology where defined.

Parameter	Symbol	Value
<i>IMU full-scale range</i>		
Accelerometer range	—	$\pm 16 \text{ g}$
Gyroscope range	—	$\pm 2000 \text{ }^\circ/\text{s}$
<i>Redundant-event suppression</i>		
Refractory threshold	κ	1 ms (1000 μs)
<i>ARC corner test</i>		
Small circle (radius 3 px), streak	$(L_{\min}, L_{\max})$	(1, 8)
Large circle (radius 4 px), streak	$(L_{\min}, L_{\max})$	(2, 14)
<i>Spatiotemporal clustering</i>		
Cluster radius (x, y)	r_x, r_y	18, 18 px
Cluster timeout	r_t	10 ms
Minimum cluster size	$n_{\min}$	10
Centroid EMA factor	α	0.2
<i>Constant-velocity Kalman tracker</i>		
Association gate distance	d_g	20 px
Max. missing interval	$t_{\text{miss,max}}$	80 ms
Process noise (position)	q	1.0
Measurement noise	r	4.0

~ 14.5 ms). All runs replay a single continuous raw-event recording (37 s, GenX320 at 320×320) of a similar hover scene as Figure 4, captured without the rest of the pipeline running, through a offline reimplement of the on-board front-end. There is one deliberate difference compared to the the flown system; confirmed tracks are counted directly after the tracker’s birth test ($n_{\text{birth}} = 2$ associations) with no further admission filter, so that the metrics respond to the parameter under study rather than to the admission threshold. These statistics are therefore not comparable to the flight statistics of Figure 5, where tracks are counted only after EKF admission (96 associations) on different flights; in addition, lifetimes here are split at internal observation gaps, so a re-acquired feature is not counted as one long track.

For simplicity we report the same metrics per varied parameter, stacked on a shared parameter axis. *Feature amount* is the number of confirmed tracks, a quantity measure. *Median track lifetime* is the median continuous observation span of a track; persistent features give the estimator long temporal baselines for triangulation and inverse-depth convergence, so this is the primary quality proxy. *Track jitter* is the median over tracks of the RMS second difference of the image position; the second difference cancels constant image velocity, leaving the high-frequency positional noise the constant-velocity tracker must damp, i.e. the front-end’s contribution to the measurement noise seen by the filter (the flown measurement-noise setting, $r = 4 \text{ px}^2$, is of the same order as the $\sim 4 \text{ px}$ jitter measured here). These are front-end proxies: they do not directly measure state-estimation accuracy. The numbers derive from one recording of one scene, so values guide relative comparisons rather than serve as absolute constants.

We set two reference timescales that recur throughout. The wing-beat fundamental is $f_{\text{flap}} = 12\text{--}17 \text{ Hz}$ (period $T_{\text{flap}} \approx 59\text{--}83 \text{ ms}$). A dominant component of the event stream, however, sits at $2f_{\text{flap}}$ (period $T_{\text{osc}} \approx 29\text{--}42 \text{ ms}$); the yaw-rate magnitude peaks twice per stroke and the aerodynamic force scales with velocity squared (Appendix A).

TABLE IX: Effect of the ARC* corner filter on the three front-end metrics. Fewer but longer-lived, slightly less jittery tracks.

Metric	ARC* off	ARC* on
Feature count	500	372
Median track lifetime [s]	0.33	0.42
Track jitter [px]	4.05	3.97

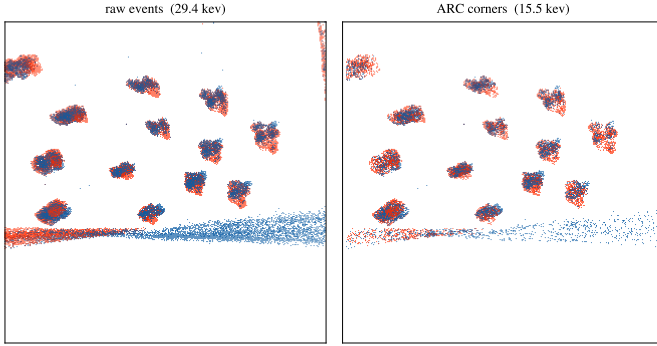


Fig. 23: 100 ms of accumulated events before (left) and after (right) the ARC* corner test (ON events red, OFF events blue). Blob features pass nearly intact; the floor band and pole edges are thinned away.

ARC* on/off

Enabling the corner test raises the median track lifetime by +26% ($0.33 \rightarrow 0.42$ s) and slightly lowers jitter ($4.05 \rightarrow 3.97$ px) while reducing the track count ($500 \rightarrow 372$; Table IX). This is the intended behavior: ARC* keeps only events on a contiguous arc of recent surface activity (57% of the suppressed events on this recording), discarding edge and flat-region events that would otherwise seed short-lived, poorly localized clusters. Figure 23 makes the mechanism visible: over the same 100 ms of events, the blob features pass the corner test nearly intact while the extended edges of the arena floor and poles are thinned away, exactly the structures that spawn unstable clusters. Two caveats qualify the gain. First, compute: the corner test costs a fixed ~ 36 surface lookups per event, paid on every suppressed event, and is the dominant on-board cost (Table IV); the 43% event reduction is only recovered downstream in clustering. Second, the size of the benefit is expected to be scene-dependent: the corner filter pays off when raw clustering is noisy, and a scene that already yields long, stable clusters gains less. It did not degrade tracking in any regime tested, so keeping it enabled is safe; the decisive test, left open, is repeating the on/off comparison on end-to-end state-estimation accuracy.

Redundant-event suppression threshold κ : At the operating point $\kappa = 1$ ms only 0.2% of events are removed and all three metrics are flat. Meaningful removal begins near T_{osc} ($\kappa = 10$ ms removes 19%) with lifetime unchanged up to $\kappa = 100$ ms (30% removed). The parameter is thus a throughput knob and the operating point is conservative: because κ gates how many events reach the ARC* circle test, the dominant on-board cost, raising it toward T_{osc} would cut that cost nearly proportionally at no measured quality loss (Figure 25 shows

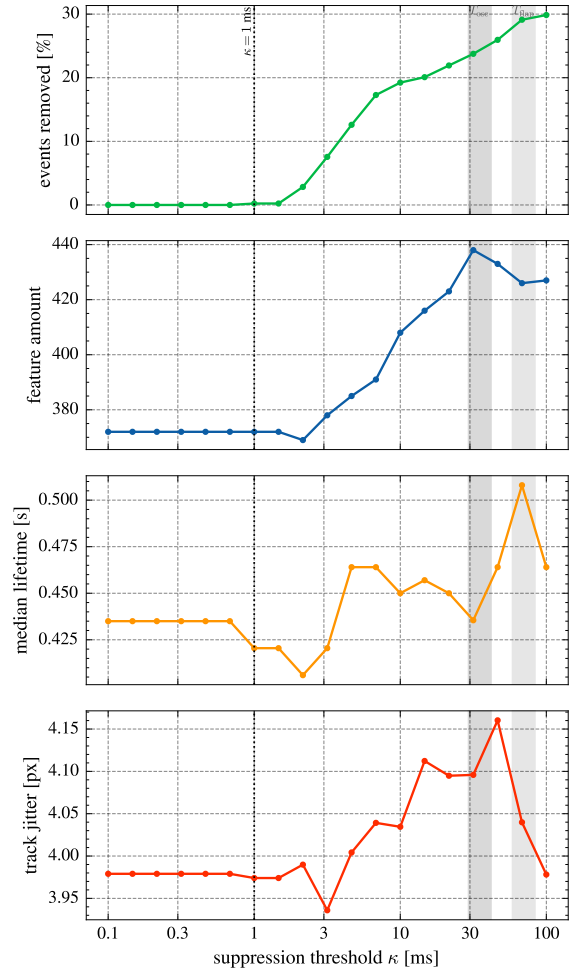


Fig. 24: Suppression threshold κ . Events removed plus the three front-end metrics; shaded bands mark T_{osc} and T_{flap} ; the operating point $\kappa = 1$ ms removes almost nothing.

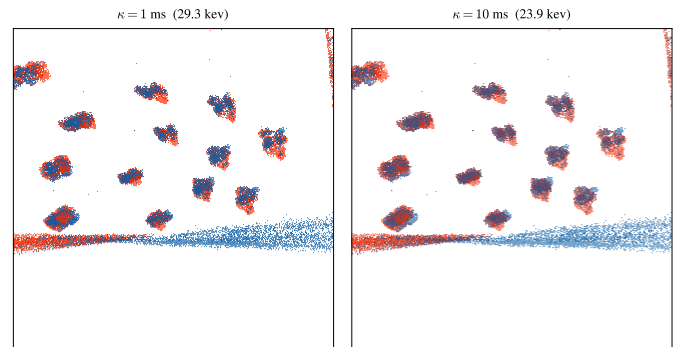


Fig. 25: The same 100 ms of suppressed events at $\kappa = 1$ ms (left) and $\kappa = 10$ ms (right). The removal is spatially uniform; feature structure is untouched.

the thinning is spatially uniform rather than concentrated on features). Beyond T_{osc} it would start erasing the slower edges.

Arc-length bound L_{max}

L_{max} is the ARC* permissiveness knob: it caps the length of the contiguous arc admitted as a corner, so a small value

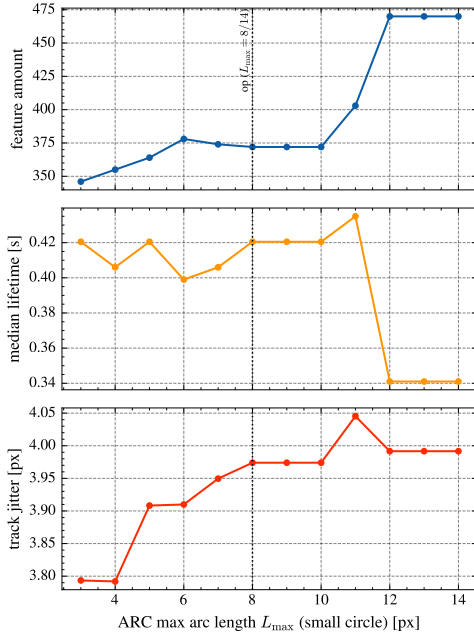


Fig. 26: ARC* arc-length bound. A wide flat plateau; quality degrades only when the bound spans the whole circle.

keeps only sharp corners and a large value also admits flatter edges. The response is a flat plateau for $L_{\max} = 4\text{--}10$ on the small circle and degrades only at 12, where the scaled large-circle bound spans the whole 20-point circle and admits non-corners (count jumps, lifetime drops; Figure 26). The operating point 8/14 sits safely inside the plateau: corner strictness is a non-critical parameter with a wide safe range.

Temporal window r_t

With a wider clustering window ($4\times$ the cycle period), r_t shows a clear knee (Figure 27): as it grows, the feature count falls and the median lifetime steps up, both saturating at $T_{\text{osc}} \approx 29\text{--}42$ ms. The knee is physically meaningful: one event-oscillation period is the span needed to accumulate a coherent blob without merging across successive flapping phases, so r_t is governed by the flapping frequency. Larger r_t is not free, however. Count and lifetime cannot distinguish clusters that live longer because they are coherent from clusters that live longer because they merge, and a cluster carries no motion model: past an activity gap the feature reappears displaced, so a surviving cluster follows it with a jumping centroid, whereas the constant-velocity tracker bridges the same gap smoothly by prediction. Hand tuning surfaced exactly this: at larger timeouts, cluster centroids jumped and neighboring clusters merged wrongly, so the flown value was set below the knee, keeping each cluster to a single coherent burst and leaving the smoothing to the tracker. The remaining head-room toward T_{osc} makes r_t a candidate for flapping-frequency-aware tuning.

Spatial radius $r_x = r_y$

The clustering radius sets the image-plane scale of a feature. The count is U-shaped with a minimum near 22–30 px, the

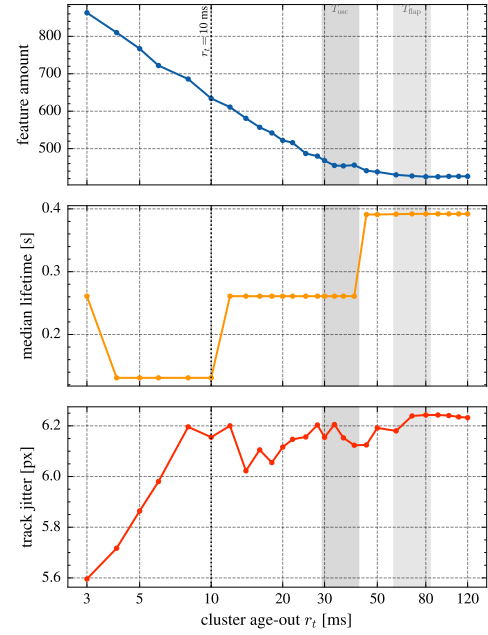


Fig. 27: Cluster age-out r_t , swept with a widened clustering window so the parameter binds. Count and lifetime saturate at the event-oscillation period $T_{\text{osc}} = 1/(2f_{\text{flap}})$ (shaded).

median lifetime peaks around 30 px (0.64 s versus 0.42 s at the 18 px operating point), and jitter grows with radius (Figure 28). The mechanism is a clear trade-off: too small a radius fragments one feature into several short-lived sub-clusters (high count, low lifetime); too large a radius merges neighboring features so the centroid hops between them (low count, high jitter). This was also the parameter with a large visible impact during the on-board manual tuning, and the sweep confirms it is the primary scale knob of the front-end. Theoretically, its optimum is not a constant of the algorithm but of the scene and motion: the radius should match the image-plane extent of a feature’s event blob, which scales with the physical feature size, with inverse depth, and with the flap-induced image-motion amplitude. Nearer surfaces, larger features, and larger self-motion favor a larger radius. The radius is therefore the strongest case for depth-adaptive tuning, though its optimum also tracks the body-oscillation amplitude.

Support threshold

A cluster is emitted only once it collects a minimum number of events. Since that count scales with cluster area, it is reported as the dimensionless density $s = n_{\min}/(r_x r_y)$, which decouples it from the radius. The trade-off is clear (Figure 29): raising s trades quantity for longer lifetime (count $647 \rightarrow 206$, median lifetime $0.35 \rightarrow 0.85$ s), because a higher value admits only well-populated clusters. The operating point ($n_{\min} = 10$, $s = 0.031$) sits at the elbow on the conservative side; there is head-room to raise it if the estimator prefers fewer, longer-lived features.

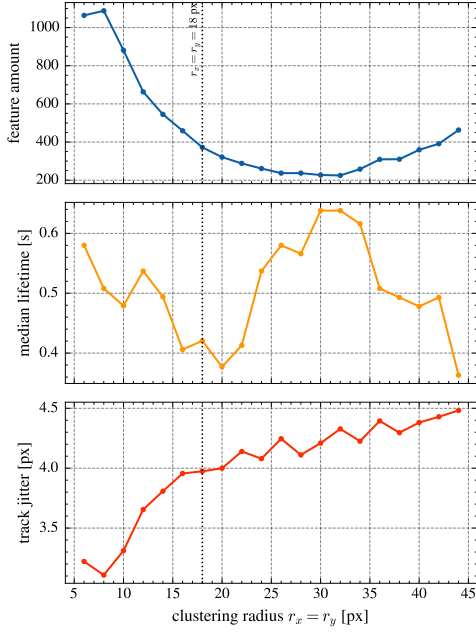


Fig. 28: Spatial clustering radius $r_x = r_y$. A merge/split trade with a lifetime optimum near 30 px on this scene; the optimum tracks scene depth and feature size.

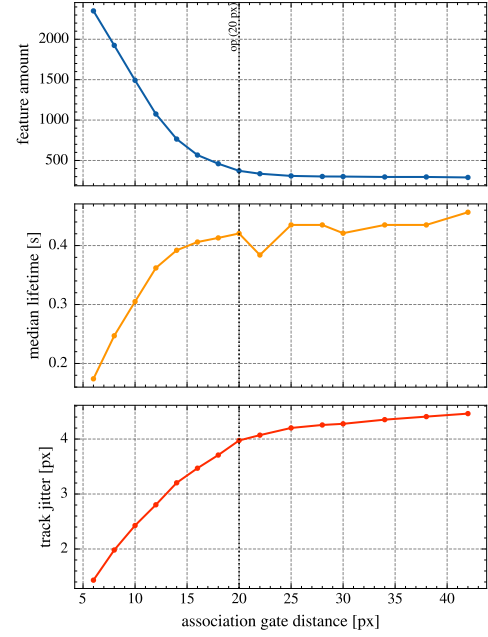


Fig. 30: Association gate distance. Small gates fragment tracks into many short-lived re-births; the operating point sits at the lifetime elbow.

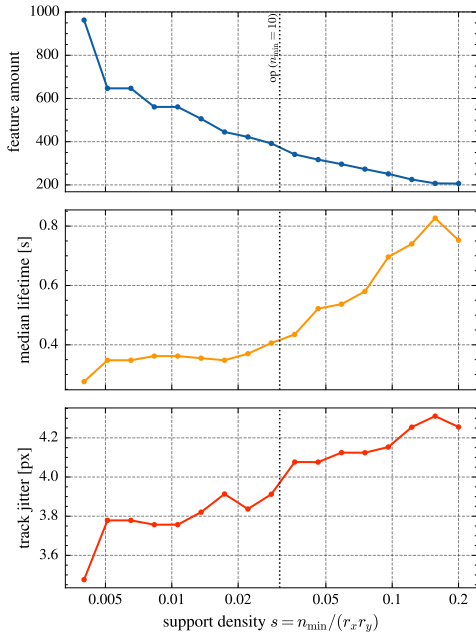


Fig. 29: Minimum cluster support as a dimensionless density s . A monotone quantity-for-persistence trade; the operating point sits at the elbow.

Association gate

A detection is bound to a track only within the gate distance of the predicted position, so the distance must cover a feature's image displacement between cycles, at hover dominated by the flap-induced self-motion. Below the operating point the gate fragments tracks: a moving feature escapes the gate and creates a new track, giving 1924 tracks at 8 px with jitter as low

as 2 px because only near-static features survive (Figure 30). Lifetime rises and plateaus at 16–20 px while the count falls sharply. The 20 px operating point sits at the elbow. The gate is coupled to the centroid jitter (it must exceed a few times the measurement noise), and it scales with inverse depth and flap amplitude. It is also a parameter that could benefit from flapping-informed tuning.

Track-death timeout

A track is dropped after *max_missing* without a matched detection. The track must bridge the brief drops in event activity at oscillation reversal, where the image velocity, and thus the event rate, is lower; these recur at T_{flap} . The trade-off is clear (count 832 $\rightarrow$ 303, lifetime 0.02 $\rightarrow$ 0.94 s; Figure 31). A longer timeout yielding fewer tracks seems counter-intuitive until one notes that at short timeouts the same physical feature is killed and re-born repeatedly; at 5 ms (below one cycle period, so a single missed cycle is fatal) the tracker logs 832 “features” with a 15 ms median lifetime, mostly re-births of the same handful of blobs. Consistently, jitter is highest at short timeouts (4.9 px, dominated by immature re-born tracks whose velocity estimate has not converged) and settles once tracks persist. The 80 ms operating point coincides with T_{flap} , i.e. it bridges one wing-beat. This is an expected and explainable limit: beyond T_{flap} a track only survives on constant-velocity prediction without new measurements, possibly adding drift to the estimator. The timeout is thus set by the flapping frequency, and larger is not necessarily better.

Centroid smoothing α

The cluster centroid is an exponential moving average of its events, $x_c += \alpha(x_e - x_c)$, a per-event low-pass. Smaller

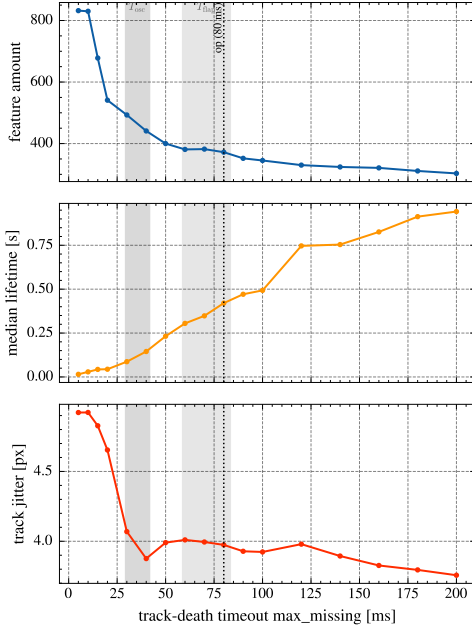


Fig. 31: Track-death timeout. Short timeouts inflate the count with re-births of the same features; the operating point bridges one wing-beat (T_{flap} , shaded).

α smooths hard, rejecting flap-induced jitter at the cost of lag; $\alpha = 1$ follows the last event with no smoothing. As expected, jitter grows with α ($3.76 \rightarrow 4.45$ px), and the count grows as well ($321 \rightarrow 620$), so the smoothing trades feature count for measurement quality (Figure 32). Given that the tracker already low-passes downstream, one may ask whether the EMA is redundant. It is not, because the two filters act on different timescales: the tracker filters across cycles at the loop rate, whereas the EMA filters within a cluster at the event rate, setting the quality of the single measurement the tracker receives. Removing it confirms this design choice: $\alpha = 1$ raises jitter by 12%, and the degenerate $\alpha = 0$ (centroid frozen at the seed event) costs 28% of the median lifetime because a frozen centroid cannot follow the blob across the cycle. The 0.2 operating point sits on the smooth side of a small optimum near $\alpha \approx 0.05\text{--}0.1$.

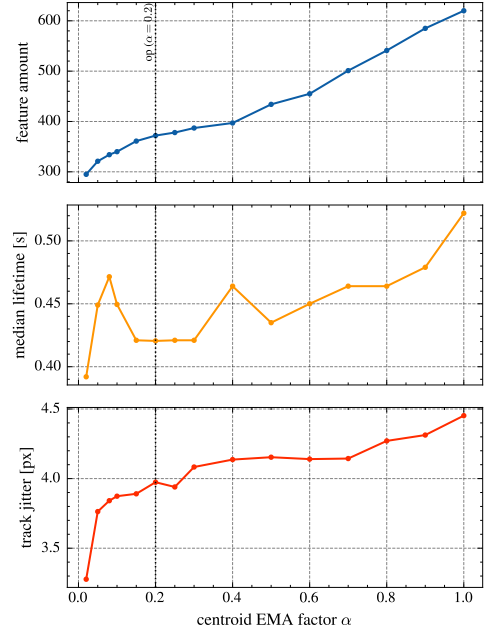


Fig. 32: Centroid EMA factor α . Jitter grows with α ; the operating point sits on the smooth side of a small optimum.

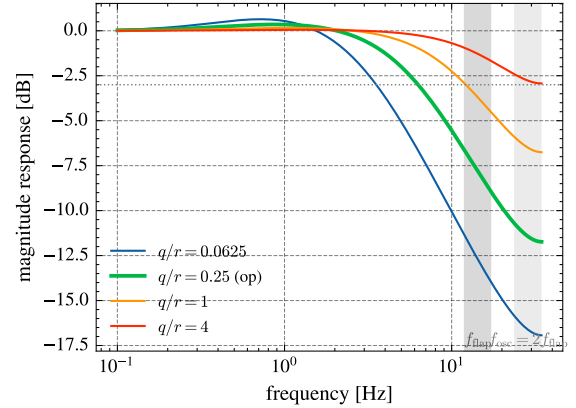


Fig. 33: Analytical frequency response of the constant-velocity tracker for several noise ratios q/r . The operating point passes body motion and damps the flapping bands (shaded).

Tracker bandwidth q/r

The constant-velocity Kalman tracker is a linear time-invariant low-pass from measured to estimated position; the process-to-measurement noise ratio q/r sets its corner frequency, with the loop period (~ 14.5 ms) as sampling time. At the operating point $q/r = 0.25$ the -3 dB corner is ~ 5 Hz (Figure 33): the filter passes the body dynamics (< 2 Hz) while damping the flap self-motion by ~ 8 dB at f_{flap} and ~ 11 dB at $2f_{\text{flap}}$. The tracker is thus a second low-pass filter complementing the centroid EMA, and its useful setting is defined relative to the flapping frequency: the corner must sit above the body-motion band and below f_{flap} .

C. Synthesis

The recurring theme is that several optima are not properties of the algorithm but of the platform. The temporal window r_t and the death timeout are, as expected, set by the flapping frequency (T_{osc} and T_{flap} respectively); the tracker and centroid bandwidths are defined relative to the flap band; and the spatial radius and association gate scale with scene depth and flapping-generated image-motion amplitude. In this sense the front-end is conservatively tuned: each value is defensible and none sits at a poor operating point, yet the same analysis shows that a flap- and depth-aware controller, adapting r_x, r_y and the gate to the estimated depth and r_t and max_missing to the measured beat frequency, is the natural route to more and longer-lived features without hand-tuning per flight.

Two main limitations bound these conclusions. First,

the analysis is around a single operating point, while the parameters are demonstrably coupled (radius–support, κ –support, gate–loop–rate, EMA–tracker bandwidth). The natural next step is a joint sweep of the front-end, ideally scored not on these proxy metrics but on end-to-end state-estimation accuracy, as was done for the estimator parameters (Appendix C). Second, all figures derive from one scene; the parameters flagged as scene-dependent (r_x, r_y , the gate) should be re-swept at other depths before their optima are taken as general.

Back-end ablations

Depth-prior fallback: Under weak near-hover parallax the metric scale partly follows the fixed depth prior (Section V). To isolate its contribution, the tuning flights are replayed with the fallback inverse depth ρ_{fb} swept from 0.1 to 0.9 m^{-1} (assumed depths 10–1.1 m) while all other parameters stay at the tuned operating point (Figure 34). Accuracy degrades steeply only when the prior places features far beyond the true scene: at $\rho_{fb} = 0.1$, an assumed depth of 10 m for a 1.5–4 m scene, the per-axis position RMSE rises to 2–3 $\times$ its default value. Above $\rho_{fb} \approx 0.3$ accuracy is nearly constant across the geometrically plausible depth range: per-axis position RMSE varies by roughly 0.1 m over the remaining sweep, and roll and pitch are nearly flat. The flown value of 0.28, fixed from the test-area geometry before tuning, lies near the edge of this near-constant range rather than at its most accurate point, consistent with a prior chosen from geometry rather than optimized for accuracy. Within the plausible range the exact value is thus not critical; only a prior far outside the true scene depth harms the estimate.

Depth-initialization tightness: New features enter the filter with a relative inverse-depth uncertainty of $\sigma_\rho/\rho = 9.5 \times 10^{-4}$ (Table VII), tight enough that their depths stay near the initialization. Because this parameter was itself part of the tuning search, we sweep it to check that the reported accuracy does not hinge on the tuned value: the tuning flights are replayed with the initialization std scaled from $0.1\times$ to $1000\times$ the adopted value ($\sigma_\rho/\rho \approx 10^{-4}$ to 0.95), all other parameters fixed (Figure 35). Accuracy is unchanged over $0.1\times$ – $3\times$ (mean ATE 0.75–0.76 m, identical to the reported operating point). From $10\times$ the mean degrades mildly (≈ 1.0 m at $10\times$ – $30\times$, driven by a single unstable flight), and once the initialization approaches uninformative ($\gtrsim 100\times$) several flights diverge, two failing outright at $1000\times$, while the median flight degrades only gradually. The adopted tightness thus sits on a plateau spanning more than an order of magnitude in either direction, so the exact value inherited from tuning is not crucial for the reported accuracy; the breakdown at near-uninformative initializations conversely confirms that parallax alone cannot establish feature depth in this regime.

Thrust (accelerometer) input: The reported configuration does not feed the measured body- z accelerometer into the prediction; instead it assumes thrust balances gravity along the vertical axis (Section IV). To assess this choice, all tuning flights are replayed with a variant that inserts the measured body- z specific force into the velocity prediction,

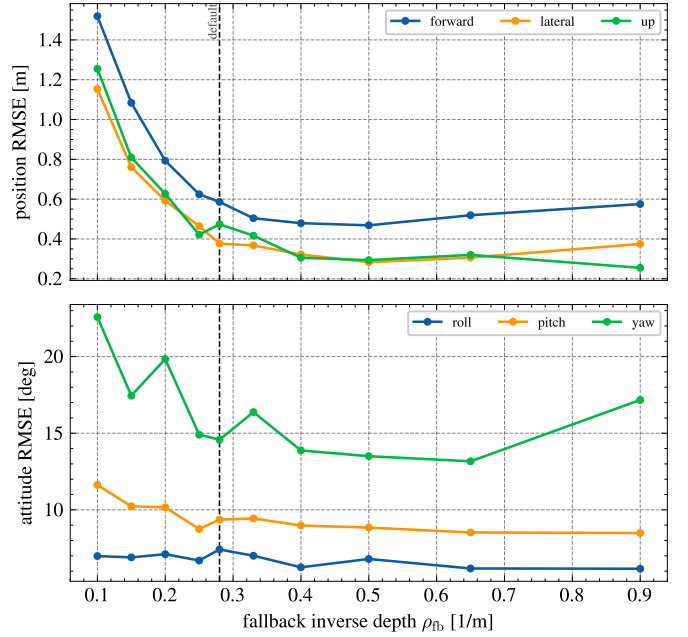


Fig. 34: Depth-prior fallback sweep: per-axis position RMSE (top) and attitude RMSE (bottom) as the fallback inverse depth ρ_{fb} is varied, with all other parameters at the tuned operating point. The dashed line marks the flown default (0.28 m^{-1}). Accuracy degrades steeply only for priors far beyond the true scene depth and is flat across the plausible range.

each configuration tuned independently (Appendix C). The comparison is scored on the vertical velocity, the axis most affected by this choice. The thrust-free configuration is more accurate on 12 of the 13 flights: inserting the thrust input raises the mean v_z RMSE from 0.23 to 0.37 m/s. Figure 36 shows a representative segment: with the thrust input the v_z estimate carries markedly larger excursions around the reference, consistent with the flapping-dominated specific force entering the prediction faster than the visual updates can correct it. This supports the thrust-free configuration used in this work.

APPENDIX E

COORDINATE FRAMES AND CONVENTIONS

All frames used in this work are right-handed; the conventions are illustrated in Figure 37.

a) World frame $\mathcal{W}$: A fixed, gravity-aligned frame with z_w pointing up (gravity acts along $-z_w$). The robot position (X, Y, Z), the world-frame velocity shown in the results, and the motion-capture ground truth are expressed in this frame. After the yaw alignment described in the results, the world X axis corresponds to the platform’s optical axis, so the results refer to world X , Y , and Z as the forward, lateral, and vertical directions, respectively.

b) Body frame $\mathcal{B}$: The body frame is fixed to the platform with its z_b axis pointing up along the thrust direction, so that at the level reset attitude z_b is parallel to the world up axis z_w and gravity acts along $-z_b$. The IMU axes coincide with the body axes (the IMU-to-body rotation is the identity in our configuration; in offline replay the recordings already

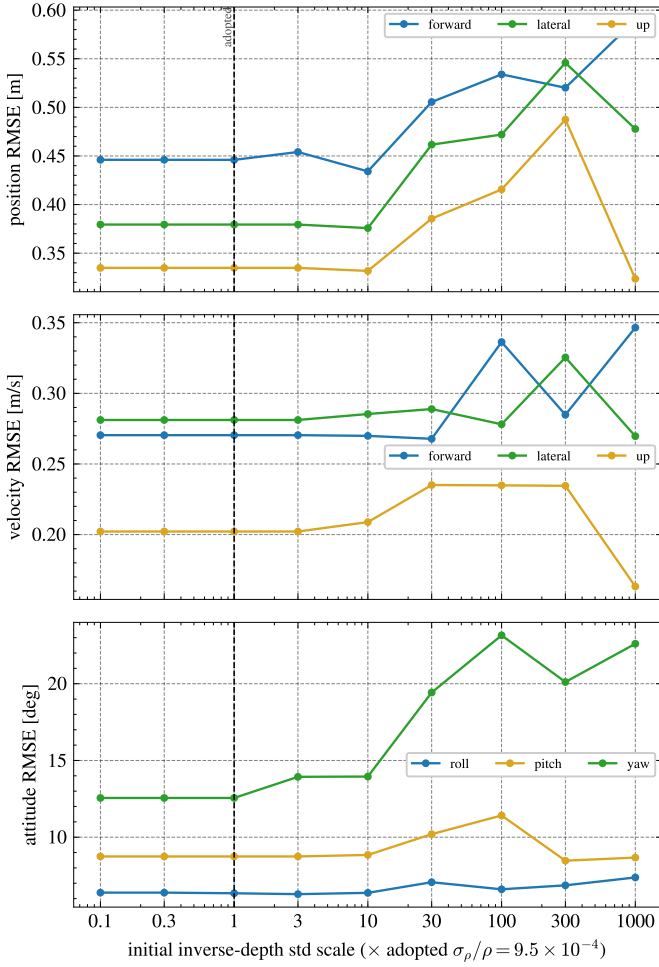


Fig. 35: Depth-initialization tightness sweep: median per-axis position, velocity, and attitude RMSE over the tuning flights as the initial inverse-depth std is scaled from $0.1\times$ to $1000\times$ the adopted $\sigma_p/\rho = 9.5 \times 10^{-4}$, all other parameters at the tuned operating point. The dashed line marks the adopted tightness and degrades once the initialization becomes nearly uninformative.

store the rotated data), so the accelerometer and gyroscope measurements, the body-frame velocity state (P_X, P_Y, P_Z) , and the small-angle attitude error are all expressed in this frame. In hover the accelerometer therefore reads a specific force of approximately $+1g$ along z_b . The remaining body axes are x_b , aligned with the camera’s viewing (optical) axis, and y_b , the lateral (image-right) direction.

c) *Camera frame C .*: The camera frame follows the standard computer-vision convention: x_c points right in the image, y_c down in the image, and z_c forward along the optical axis (depth).

d) *Attitude and Euler angles.*: The attitude is the rotation $\mathbf{R}$ from $\mathcal{B}$ to $\mathcal{W}$, maintained internally as a quaternion with a small-angle error state (Appendix G-A). The error states (D_0, D_1, D_2) , and the roll, pitch, and yaw angles reported in the results, are defined about the body axes: roll $\phi = D_0$ about x_b (the optical/viewing axis), pitch $\theta = D_1$ about y_b (lateral),

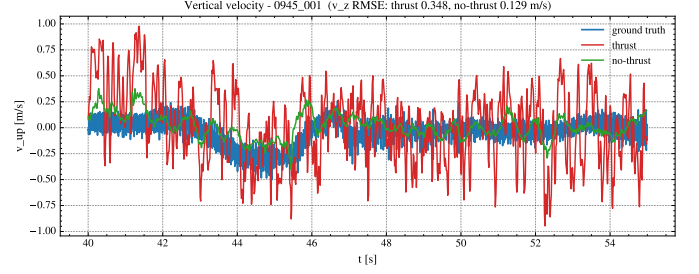


Fig. 36: Vertical (up) velocity over a representative segment of flight G (40–55 s): motion-capture reference versus the estimate with and without the measured body- z (thrust) specific force in the prediction.

and yaw $\psi = D_2$ about z_b (vertical).

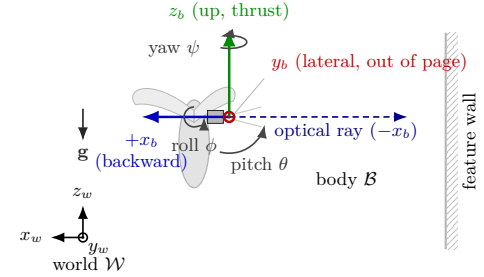


Fig. 37: Coordinate conventions, drawn at the level (hover) pose where thrust opposes gravity. World frame $\mathcal{W}$ is fixed and gravity-aligned with z_w up. Body frame $\mathcal{B}$ has z_b up along the thrust axis, y_b lateral (out of the page), and x_b horizontal with its positive direction pointing backward, away from the wall; the camera views along $-x_b$ (dashed ray) toward the feature wall, so image-up is along $+z_b$ and image-right along $+y_b$ (related to $\mathcal{B}$ by the fixed rotation $\mathbf{R}_{cb}$). Roll, pitch, and yaw are right-handed rotations about $+x_b$, $+y_b$, $+z_b$; because $+x_b$ points backward, a positive roll raises the right-hand side. The world frame is drawn coinciding with the body frame at this reset pose ($x_w = x_b$, $y_w = y_b$, $z_w = z_b$); the camera, and thus the platform’s viewing direction, looks along $-x_b$ toward the wall.

APPENDIX F ARC FILTERING AND CLUSTERING DETAILS

A. Event surface representation

The event camera stream is represented using two polarity-separated timestamp surfaces:

$S_p(x, y)$: reference surface of accepted events for polarity p ,

$L_p(x, y)$: latest-event surface for polarity p .

For an incoming event $e_k = (t_k, x_k, y_k, p)$ the acceptance rule is:

accept if $t_k > L_p(x_k, y_k) + \kappa$ or $L_{-p}(x_k, y_k) > L_p(x_k, y_k)$.

If accepted:

$$\begin{aligned} S_p(x_k, y_k) &\leftarrow t_k, \\ L_p(x_k, y_k) &\leftarrow t_k. \end{aligned}$$

If rejected:

$$L_p(x_k, y_k) \leftarrow t_k.$$

B. ARC* event test

For each accepted event the ARC* filter evaluates two circular masks with radius parameters r_s and r_ℓ . Events within r_ℓ pixels of the image border are rejected before the test so that both masks remain fully defined. Define the sampled reference timestamps on a circle as

$$C = \{S_p(x_k + \Delta x_i, y_k + \Delta y_i)\}_{i=1}^N.$$

Let

$$i_{\max} = \arg \max_i C_i, \quad C_{\max} = C_{i_{\max}}.$$

The newest-arc A_{new} is the contiguous segment of indices around $i_{\max}$ whose timestamps remain recent relative to the candidate arc support. The corner decision is then:

$$L_{\min} \leq |A_{\text{new}}| \leq L_{\max} \quad \text{or} \quad L_{\min} \leq N - |A_{\text{new}}| \leq L_{\max}.$$

The event passes ARC only if the condition holds on both the small and large circle masks.

C. Spatiotemporal clustering

Each active cluster c stores:

- centroid (x_c, y_c) ,
- last timestamp $t_{c,\text{last}}$,
- count n_c ,
- bounding box $(x_{\min}, x_{\max}, y_{\min}, y_{\max})$,
- timestamp accumulator $\sum t_i$.

A cluster is deactivated if

$$t_k - t_{c,\text{last}} > r_t.$$

Candidate clusters satisfy the gate:

$$\begin{aligned} |x_k - x_c| &\leq r_x, \\ |y_k - y_c| &\leq r_y. \end{aligned}$$

The normalized elliptical distance used to select the best association is

$$d(c, e_k) = \frac{(x_k - x_c)^2}{r_x^2} + \frac{(y_k - y_c)^2}{r_y^2}.$$

If several clusters pass the gate, they are first merged into the best candidate by event-count-weighted centroid averaging and bounding-box union. The chosen cluster is then updated as:

$$\begin{aligned} x_c &\leftarrow x_c + \alpha(x_k - x_c), \\ y_c &\leftarrow y_c + \alpha(y_k - y_c), \\ t_{c,\text{last}} &\leftarrow t_k, \\ n_c &\leftarrow n_c + 1, \\ x_{\min} &\leftarrow \min(x_{\min}, x_k), \\ x_{\max} &\leftarrow \max(x_{\max}, x_k), \\ y_{\min} &\leftarrow \min(y_{\min}, y_k), \\ y_{\max} &\leftarrow \max(y_{\max}, y_k). \end{aligned}$$

A cluster is emitted as a detection if

$$n_c \geq n_{\min}.$$

The cluster timestamp can be represented as the mean

$$t_c = \frac{1}{n_c} \sum_{i=1}^{n_c} t_i.$$

D. Constant-velocity Kalman tracking

Each image-plane track is modeled by the state

$$\mathbf{x}_k = [x_k, y_k, v_{x,k}, v_{y,k}]^\top,$$

with constant-velocity transition

$$\mathbf{x}_{k|k-1} = \mathbf{F}_k \mathbf{x}_{k-1|k-1}, \quad \mathbf{F}_k = \begin{bmatrix} 1 & 0 & \Delta t_k & 0 \\ 0 & 1 & 0 & \Delta t_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The covariance prediction is

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^\top + \mathbf{Q},$$

where

$$\mathbf{Q} = \text{diag}(q, q, 10q, 10q).$$

The measurement model observes only position:

$$\mathbf{z}_k = \mathbf{H} \mathbf{x}_k + \mathbf{v}_k, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad \mathbf{R} = r \mathbf{I}_2.$$

For a detection $\mathbf{z}_k = [x_k^{\text{det}}, y_k^{\text{det}}]^\top$, the standard Kalman innovation is

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H} \mathbf{x}_{k|k-1},$$

with innovation covariance and gain

$$\begin{aligned} \mathbf{S}_k &= \mathbf{H} \mathbf{P}_{k|k-1} \mathbf{H}^\top + \mathbf{R}, \\ \mathbf{K}_k &= \mathbf{P}_{k|k-1} \mathbf{H}^\top \mathbf{S}_k^{-1}. \end{aligned}$$

The corrected state and covariance become

$$\begin{aligned} \mathbf{x}_{k|k} &= \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{y}_k, \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}) \mathbf{P}_{k|k-1}. \end{aligned}$$

This low-order tracker acts as a temporal regularizer on the event clusters, suppressing high-frequency image jitter while remaining cheap enough for on-board real-time execution.

E. Data association and track management

At each detection time, all active tracks are first propagated using the elapsed interval Δt_k , derived from successive detection timestamps. Candidate detection-track pairs are accepted by a spatial gate

$$\|\mathbf{z}_k - \hat{\mathbf{z}}_k\|_2 < d_g,$$

where d_g is a fixed Euclidean gating threshold. All admissible pairs are sorted by distance and assigned greedily, yielding a one-to-one matching without solving a global assignment problem.

Unmatched detections initialize new tracks. Matched tracks increment their update counters, whereas unmatched tracks

increment a missing counter and lose consecutive-hit status. A track is promoted to a confirmed feature track once it reaches a minimum number of consecutive successful associations,

$$n_{\text{consecutive}} \geq n_{\text{birth}},$$

and it is deleted if the time since its last successful update exceeds a maximum missing interval $t_{\text{miss,max}}$.

APPENDIX G

EKF DYNAMICS, COVARIANCE AND FEATURE UPDATE

A. State representation

The robot state is

$$\mathbf{x}_r = [X, Y, Z, P_X, P_Y, P_Z, D_0, D_1, D_2]^\top.$$

Here (X, Y, Z) denotes position, (P_X, P_Y, P_Z) body-frame velocity, and (D_0, D_1, D_2) the small-angle attitude-error state used for covariance propagation. The nominal attitude is maintained separately as a quaternion and its corresponding rotation matrix. Each feature is stored as a unit bearing vector $\mathbf{h}_i \in \mathbb{S}^2$ ($\|\mathbf{h}_i\| = 1$) together with a scalar depth parameter p_i , so that its world point is

$$\mathbf{X}_{w,i} = d(p_i) \mathbf{h}_i,$$

where $d(p)$ maps the parameter to the metric distance from the world origin. The inverse-depth parametrization used here sets $p = 1/d$, i.e. $\mathbf{X}_{w,i} = \mathbf{h}_i/p_i$. The three feature error states are two bearing angles (δ_1, δ_2) in an orthonormal tangent basis $\{\mathbf{t}_1, \mathbf{t}_2\}$ at $\mathbf{h}_i$ together with the depth parameter; bearing corrections move $\mathbf{h}_i$ within this tangent plane and are renormalised onto $\mathbb{S}^2$. The compact (b_x, b_y, ρ) form used in the main text is the reported projection $(h_x/h_z, h_y/h_z, 1/d)$ of this representation. The full filter state stacks the robot state with the active feature states,

$$\mathbf{x} = [\mathbf{x}_r^\top, \mathbf{f}_1^\top, \dots, \mathbf{f}_M^\top]^\top.$$

B. Robot dynamics and prediction

To keep the EKF suitable for the embedded microcontroller, the predictor uses a low-order inertial model. The in-flight predictor retains the dominant translational, gravity, rotational, and drag effects while avoiding a higher-dimensional aerodynamic model. In particular, it propagates attitude from the gyroscope, and the only specific-force term in the translational model is a_z on the body z axis, the vertical (thrust) axis of Appendix E. In the reported thrust-free configuration this term is not taken from the accelerometer but fixed to the gravitational magnitude, $a_z = g$, under the near-hover assumption that thrust balances gravity (Section IV). No measured accelerometer channel enters the translational prediction: the body- x and body- y axes (the optical/viewing and lateral axes) as well as the body- z axis are driven by gravity projection, rotational coupling, linear drag, and the fixed thrust term alone.

With the thrust term a_z (fixed to g in the reported configuration), drag coefficients (b_x, b_y, b_z) , and center-of-pressure offset contribution

$$\boldsymbol{\omega} \times \boldsymbol{\ell}_{\text{CoP}} = [o_x, o_y, o_z]^\top,$$

the body-frame position increment becomes

$$\begin{aligned} \Delta x &= p_x dt, \\ \Delta y &= p_y dt, \\ \Delta z &= p_z dt + \frac{1}{2} a_z dt^2, \end{aligned}$$

followed by gravity-compensated integration in the world frame:

$$\mathbf{r}_{k|k-1} \leftarrow \mathbf{r}_{k-1|k-1} + \mathbf{R} \Delta \mathbf{p} - \frac{1}{2} g dt^2 \mathbf{e}_z.$$

The body-frame velocity update is

$$\begin{aligned} p_x &\leftarrow p_x + dt (\omega_z p_y - \omega_y p_z - g R_{2,0} - b_x p_x + b_x o_x), \\ p_y &\leftarrow p_y + dt (-\omega_z p_x + \omega_x p_z - g R_{2,1} - b_y p_y + b_y o_y), \\ p_z &\leftarrow p_z + dt (a_z + \omega_y p_x - \omega_x p_y - g R_{2,2} - b_z p_z + b_z o_z). \end{aligned}$$

The predictor integrates over the true elapsed interval dt between successive prediction steps, measured from the IMU timestamps both on-board and when replaying recorded data offline. The nominal interval

$$dt = 1/f_{\text{update}}$$

is only the target loop period; it is not a fixed value substituted into the propagation. Process noise is added separately after state and covariance propagation (Appendix G-D). The quaternion update uses the gyroscope-derived small-angle increment:

$$\mathbf{d} = \frac{dt}{2} [\omega_x, \omega_y, \omega_z]^\top,$$

with quaternion increment

$$dq = \begin{bmatrix} \cos(\|\mathbf{d}\|/2) \\ \sin(\|\mathbf{d}\|/2) \frac{\mathbf{d}}{\|\mathbf{d}\|} \end{bmatrix}.$$

The attitude quaternion is updated by

$$\mathbf{q} \leftarrow dq \otimes \mathbf{q}.$$

The angular rate $\boldsymbol{\omega}$ used throughout the prediction is low-pass filtered beforehand with a Butterworth filter at a 1.9 Hz cutoff, selected by parameter sweep.

C. Linearized state transition

The prediction Jacobian is denoted

$$\mathbf{A}_k = \frac{\partial f}{\partial \mathbf{x}_r}.$$

Its block structure includes position-velocity coupling, velocity-attitude coupling, and attitude self-rotation terms. The covariance propagation is

$$\begin{aligned} \mathbf{P}_{rr,k|k-1} &= \mathbf{A}_k \mathbf{P}_{rr,k-1|k-1} \mathbf{A}_k^\top, \\ \mathbf{P}_{rf,k|k-1} &= \mathbf{A}_k \mathbf{P}_{rf,k-1|k-1}. \end{aligned}$$

The feature covariance blocks remain unchanged in the prediction step when no explicit feature process noise is modeled.

D. Process noise addition

Process noise is added as diagonal uncertainty using a continuous-time (random-walk) discretization, so that the noise parameters are interpreted as spectral densities $[\cdot/\sqrt{\text{Hz}}]$ rather than per-step standard deviations. With acceleration spectral density σ_a (per axis, $\sigma_{a,xy}$ or $\sigma_{a,z}$), velocity and position random-walk densities σ_v, σ_p , and gyro spectral density σ_g (roll/pitch or yaw) combined with the attitude density σ_{att} , the variance increments over the elapsed interval dt are

$$\begin{aligned} P_{XX}, P_{YY} &+= (\sigma_{a,xy}^2 + \sigma_v^2) \frac{dt^3}{3} + \sigma_p^2 dt, \\ P_{ZZ} &+= (\sigma_{a,z}^2 + \sigma_v^2) \frac{dt^3}{3} + \sigma_p^2 dt, \\ P_{P_X P_X}, P_{P_Y P_Y} &+= (\sigma_{a,xy}^2 + \sigma_v^2) dt, \\ P_{P_Z P_Z} &+= (\sigma_{a,z}^2 + \sigma_v^2) dt, \\ P_{D_0 D_0}, P_{D_1 D_1} &+= (\sigma_{g,\text{rollpitch}}^2 + \sigma_{\text{att}}^2) dt, \\ P_{D_2 D_2} &+= (\sigma_{g,\text{yaw}}^2 + \sigma_{\text{att}}^2) dt. \end{aligned}$$

The velocity and attitude increments scale linearly with dt , while the position increment combines the linear random-walk term $\sigma_p^2 dt$ with a $dt^3/3$ term from integrating velocity noise into position. Because the noise is specified as continuous-time spectral densities tied to the elapsed interval rather than as fixed per-step variances, the same tuned parameters can be reused across the different inference rates of flight and offline replay without rescaling.

E. Visual measurement Jacobians

Tracked image measurements are first converted from pixel coordinates to undistorted normalized image coordinates. For a feature with world point $\mathbf{X}_w$ and camera-frame point $\mathbf{X}_c = [X_c, Y_c, Z_c]^\top$,

$$\hat{\mathbf{z}} = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}.$$

The projection Jacobian is

$$\mathbf{J}_\pi = \begin{bmatrix} Z_c^{-1} & 0 & -X_c Z_c^{-2} \\ 0 & Z_c^{-1} & -Y_c Z_c^{-2} \end{bmatrix}.$$

The robot Jacobian is partitioned as:

$$\begin{aligned} \mathbf{H}_r &= [\mathbf{H}_{r,\text{pos}}, \mathbf{0}, \mathbf{H}_{r,\text{att}}], \\ \mathbf{H}_{r,\text{pos}} &= -\mathbf{J}_\pi \mathbf{R}^\top, \\ \mathbf{H}_{r,\text{att}} &= \mathbf{J}_\pi [\mathbf{X}_c]_\times, \end{aligned}$$

where $[\mathbf{X}_c]_\times$ is the skew-symmetric matrix of $\mathbf{X}_c$. The feature Jacobian is expressed in the tangent-bearing/depth coordinates (δ_1, δ_2, p) :

$$\mathbf{H}_f = \mathbf{J}_\pi \begin{bmatrix} \frac{\partial \mathbf{X}_c}{\partial \delta_1} & \frac{\partial \mathbf{X}_c}{\partial \delta_2} & \frac{\partial \mathbf{X}_c}{\partial p} \end{bmatrix},$$

with

$$\begin{aligned} \frac{\partial \mathbf{X}_c}{\partial \delta_1} &= d \mathbf{R}^\top \mathbf{t}_1, \\ \frac{\partial \mathbf{X}_c}{\partial \delta_2} &= d \mathbf{R}^\top \mathbf{t}_2, \\ \frac{\partial \mathbf{X}_c}{\partial p} &= \frac{dd}{dp} \mathbf{R}^\top \mathbf{h}, \end{aligned}$$

where $d = d(p)$ and $\{\mathbf{t}_1, \mathbf{t}_2\}$ is the tangent basis at the bearing $\mathbf{h}$. For the inverse-depth parametrization $dd/dp = -1/p^2$.

F. Feature initialization

A confirmed front-end track that is not yet represented in the filter is inserted as a new feature state. The track's current pixel position is undistorted to a normalized ray $(z_x, z_y, 1)^\top$ using the calibrated camera model, back-projected at the initial inverse depth ρ , and stored as a world-frame unit bearing

$$\mathbf{h} = \frac{\mathbf{t} + \rho^{-1} \mathbf{R} \mathbf{R}_{cb} (z_x, z_y, 1)^\top}{\|\mathbf{t} + \rho^{-1} \mathbf{R} \mathbf{R}_{cb} (z_x, z_y, 1)^\top\|},$$

with $\mathbf{R}$ the current body-to-world rotation, $\mathbf{R}_{cb}$ the fixed camera-to-body mount rotation, and $\mathbf{t}$ the current camera position in the world frame. The bearing is thus taken directly from the measurement; the inverse depth is the only tuned quantity at initialization. It is set to the median inverse depth of the converged features in the map, so a new feature inherits the scene depth already established by the filter; when fewer than three converged features qualify, a fixed fallback inverse depth of 0.28 m^{-1} is used (Section V). The initial inverse-depth standard deviation is set relative to the initialized value (σ_ρ/ρ ; Table VII).

G. EKF measurement update

The innovation covariance is

$$\mathbf{S} = \mathbf{H}_r \mathbf{P}_{rr} \mathbf{H}_r^\top + \mathbf{H}_r \mathbf{P}_{rf} \mathbf{H}_f^\top + \mathbf{H}_f \mathbf{P}_{fr} \mathbf{H}_r^\top + \mathbf{H}_f \mathbf{P}_{ff} \mathbf{H}_f^\top + \mathbf{R}.$$

a) *Innovation gating.*: Each observation is checked using a Mahalanobis gate on its normalized squared innovation,

$$d^2 = \mathbf{r}^\top \mathbf{S}^{-1} \mathbf{r},$$

and is rejected (the update is skipped, leaving $\mathbf{x}$ and $\mathbf{P}$ unchanged) whenever

$$d^2 > \gamma.$$

Since $d^2 \sim \chi_2^2$ for a consistent two-dimensional residual, the threshold γ is taken from a χ_2^2 quantile just above the 99% point ($\chi_2^2(0.99) = 9.210$); the value used is listed in Table VII. A rejected feature is not removed from the map and stays available for subsequent updates. The gate is held open (disabled) for a short warm-up interval after (re)initialization so the filter can converge before outlier rejection is enforced.

Then the Kalman gains are

$$\begin{aligned} \mathbf{K}_r &= (\mathbf{P}_{rr} \mathbf{H}_r^\top + \mathbf{P}_{rf} \mathbf{H}_f^\top) \mathbf{S}^{-1}, \\ \mathbf{K}_f &= (\mathbf{P}_{fr} \mathbf{H}_r^\top + \mathbf{P}_{ff} \mathbf{H}_f^\top) \mathbf{S}^{-1}. \end{aligned}$$

The state updates are

$$\begin{aligned} \mathbf{x}_r &\leftarrow \mathbf{x}_r + \mathbf{K}_r \mathbf{r}, \\ \mathbf{f} &\leftarrow \mathbf{f} + \mathbf{K}_f \mathbf{r}. \end{aligned}$$

The covariance updates are

$$\begin{aligned} \mathbf{P}_{rr} &\leftarrow \mathbf{P}_{rr} - \mathbf{P}_{rr} \mathbf{H}_r^\top \mathbf{K}_r^\top - \mathbf{P}_{rf} \mathbf{H}_f^\top \mathbf{K}_r^\top, \\ \mathbf{P}_{rf} &\leftarrow \mathbf{P}_{rf} - \mathbf{P}_{rr} \mathbf{H}_r^\top \mathbf{K}_f^\top - \mathbf{P}_{rf} \mathbf{H}_f^\top \mathbf{K}_f^\top, \\ \mathbf{P}_{ff} &\leftarrow \mathbf{P}_{ff} - \mathbf{P}_{fr} \mathbf{H}_r^\top \mathbf{K}_f^\top - \mathbf{P}_{ff} \mathbf{H}_f^\top \mathbf{K}_f^\top. \end{aligned}$$

For other active feature blocks $j \neq i$, the cross-covariances are updated as

$$\mathbf{P}_{rfj} \leftarrow \mathbf{P}_{rfj} - \mathbf{K}_r (\mathbf{H}_r \mathbf{P}_{rfj}).$$

H. Gravity-reference attitude update

Once per prediction cycle in flight, the accelerometer direction is used as a weak roll/pitch reference. The normalized measured specific force in the body frame,

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{\|\mathbf{a}\|},$$

is compared against the predicted up direction expressed in the body frame, which is the third row of the body-to-world rotation matrix,

$$\hat{\mathbf{g}} = [R_{2,0}, R_{2,1}, R_{2,2}]^\top, \quad \mathbf{r}_g = \hat{\mathbf{a}} - \hat{\mathbf{g}}.$$

The measurement Jacobian with respect to the attitude-error state is the skew-symmetric matrix

$$\mathbf{H}_g = [\hat{\mathbf{g}}]_\times,$$

acting only on the attitude-error columns of the state. This Jacobian is rank-2: rotation about the gravity direction (yaw) is unobservable from the gravity reference. The innovation covariance is

$$\mathbf{S}_g = \mathbf{H}_g \mathbf{P}_{DD} \mathbf{H}_g^\top + \sigma_{\text{ref}}^2 \mathbf{I}_3,$$

where $\mathbf{P}_{DD}$ is the attitude-error covariance block and σ_{ref}^2 the reference measurement variance, after which the gain and the state and covariance corrections follow the standard EKF form using the full cross-covariance columns. An optional magnitude gate rejects samples whose norm deviates from g by more than a tolerance, suppressing strongly non-gravitational samples during maneuvers. The measurement variance is set large, since the instantaneous accelerometer signal is dominated by thrust and flapping-induced loads; the reference only loosely bounds roll and pitch over time.

I. Measured prediction interval and rate-invariant covariance

The prediction step uses the true elapsed time dt measured between successive predictions, both on-board and in offline replay, in the state transition and the process-noise increment; the nominal $1/f_{\text{update}}$ is only the target loop rate. Combined with the continuous-time process-noise discretization of Appendix G-D, where the inserted uncertainty is tied to the elapsed interval, the accumulated covariance over a given elapsed time is independent of the achieved update rate. The same tuned noise spectral densities apply regardless of cycle-time variation, whether from on-board loop jitter or from the differing effective rates across recordings.

APPENDIX H

NOTATION AND MATRIX DIMENSIONS

- $\mathbf{P}_{rr} \in \mathbb{R}^{9 \times 9}$,
- $\mathbf{P}_{rf} \in \mathbb{R}^{9 \times 3}$,
- $\mathbf{P}_{ff} \in \mathbb{R}^{3 \times 3}$,
- $\mathbf{H}_r \in \mathbb{R}^{2 \times 9}$,
- $\mathbf{H}_f \in \mathbb{R}^{2 \times 3}$,
- $\mathbf{K}_r \in \mathbb{R}^{9 \times 2}$,
- $\mathbf{K}_f \in \mathbb{R}^{3 \times 2}$,
- $\mathbf{S} \in \mathbb{R}^{2 \times 2}$.

APPENDIX I

EVENT CAMERA SETTINGS

For all recordings the GenX320 was configured with the following sensor parameters:

Parameter	Value
DIFF_OFF	44
DIFF_ON	60
HPF	55
FO	19
REFR	0

APPENDIX J

ADDITIONAL FIGURES

Additional figures and supplemental visual material are provided here to support the main results discussed in the paper. These figures are referenced from the main text when needed.

A. Position, velocity, and attitude for flight P

For completeness, the world-frame position, velocity, and body attitude of the higher-drift held-out flight P are shown in Figures 38–40, complementing the representative flight F shown in the main text. The estimate tracks the reference over most of the recording but diverges more strongly toward the end, around the 100–115 s mark, where the position drift is most visible.

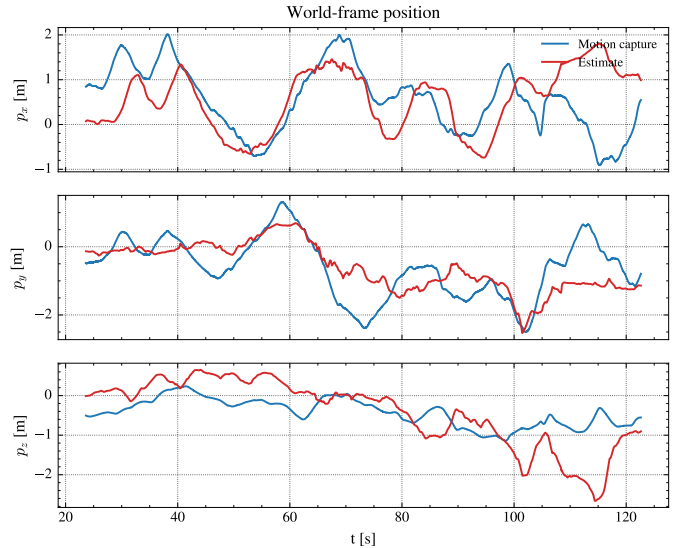


Fig. 38: World-frame position estimate for flight P compared to the motion-capture reference (estimate in red, motion capture in blue).

B. Within-flight attitude drift distribution

Figure 41 shows the distribution behind the attitude panel of Figure 9: the signed within-flight attitude drift, i.e., the 10 s window-mean of the attitude error zeroed at each flight's start, for all 10 s segments. Roll is near zero-mean ($\mu = -0.6^\circ$):

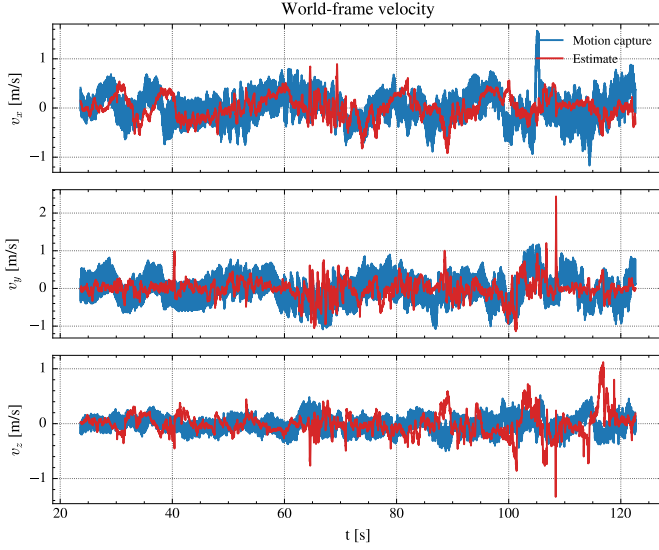


Fig. 39: World-frame velocity estimate for flight P compared to the motion-capture reference (estimate in red, motion capture in blue).

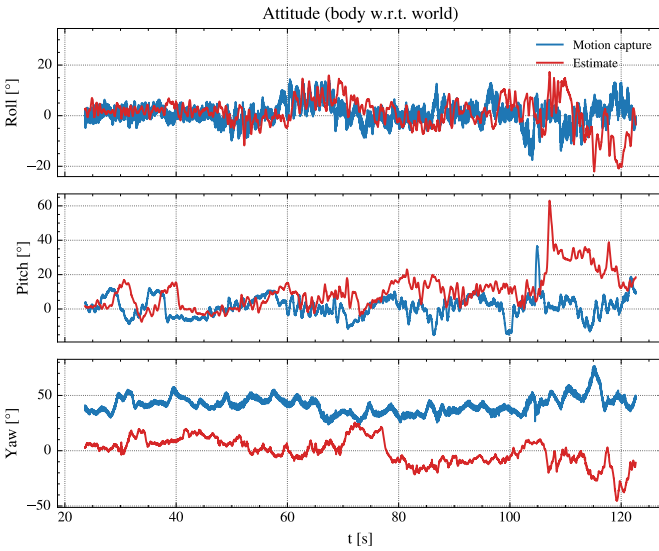


Fig. 40: Body attitude with respect to the world frame for flight P compared to the motion-capture reference (estimate in red, motion capture in blue).

the gravity reference pins it, so tilt errors flip sign and cancel rather than persist. Pitch carries a persistent positive offset ($\mu = +3.1^\circ$); the most likely cause is mechanical rather than estimation error, as the sensor board could sit pitched up by a few degrees depending on its mounting, which appears against the motion-capture reference as a constant offset and suggests that part of the reported pitch error is a calibration artifact. Yaw is zero-mean over the dataset ($\mu \approx 0^\circ$) but has by far the widest spread (standard deviation $\sim 9.9^\circ$): individual segments carry large, slowly varying heading offsets of either sign, the behavior underlying the lateral position-drift growth discussed in Section VI.

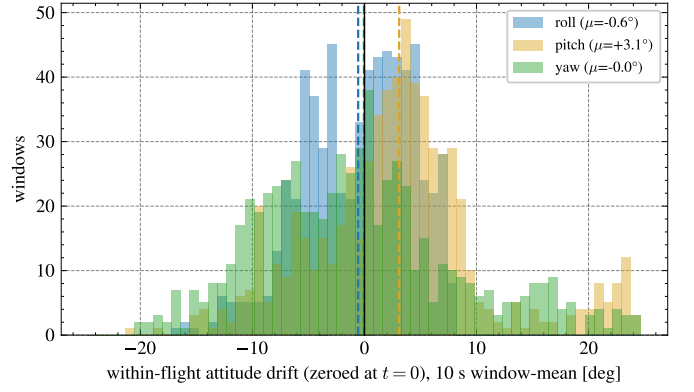


Fig. 41: Distribution of the signed within-flight attitude drift over all 10 s segments ($N = 13$ tuning flights): the window-mean attitude error, zeroed at each flight's start. Roll is near zero-mean, pitch carries a persistent $+3^\circ$ offset, and yaw is zero-mean with the widest spread.

C. Per-flight RMSE breakdown

Table X reports the RMSE of every individual flight used in the evaluation, including the three held-out flights, providing the dispersion behind the mean $\pm$ standard deviation values summarized in Table I and the generalization check of Table II.

APPENDIX K CODE AND DATA AVAILABILITY

Proposed Method and Videos

All code for this work is collected in a single repository, <https://github.com/TijnVennink/onboard-evio-fw-mav>, which is organized into subfolders: the on-board camera firmware and flight-control software, the offline replay and parameter-sweep tools used for the back-end evaluation. Videos of the on-board closed-loop hover (Section V) are included in the same repository.

Flight Datasets

The recorded flight datasets used in the evaluation are available at <http://tiny.cc/xej6101>. The mapping between the flight letters A–P used in this work and the recording names in the released datasets is included with the data (`flight_names.csv`).

Flapper Nimble+ Firmware and Default Parameters

The on-board closed-loop hover (Section V) used the stock Flapper Nimble+ flight firmware and its default controller and estimator parameters, documented by the manufacturer at https://docs.flapper-drones.eu/doku.php?id=nimbleplus:scientific_papers and defined in the Crazyflie firmware at https://github.com/bitcraze/crazyflie-firmware/blob/master/src/platform/interface/platform_defaults_flapper.h.

TABLE X: Per-flight RMSE for the 13 tuning flights and the 3 held-out flights, after the 4-DOF alignment. Flights are labeled A–M (tuning) and N–P (held-out). Position errors (x, y, z) and the total position RMSE (ATE) are in metres, the total velocity error in m/s, and the attitude errors (roll, pitch, yaw) in degrees. Here n is the number of evaluated state estimates and T the evaluated duration. The two summary rows give the tuning-set mean $\pm$ standard deviation (repeating the main-text values) and the held-out mean.

Flight	n	T [s]	Position [m]				Vel. [m/s]	Attitude [deg]		
			x	y	z	ATE	total	roll	pitch	yaw
A	2281	33	0.71	0.34	0.25	0.83	0.50	4.6	8.7	12.5
B	2713	40	0.41	0.47	0.33	0.71	0.44	5.1	6.5	26.2
C	1915	26	0.62	0.21	0.19	0.68	0.45	6.6	5.1	9.8
D	3037	40	0.24	0.14	0.20	0.34	0.43	6.1	7.6	5.1
E	3664	44	0.48	0.37	0.51	0.79	0.57	8.9	8.9	9.5
F	3662	48	0.20	0.20	0.26	0.38	0.34	4.5	5.2	6.1
G	2136	35	0.20	0.30	0.42	0.55	0.40	7.6	10.0	25.4
H	6341	111	0.45	0.84	0.51	1.08	0.48	10.6	12.7	17.5
I	6111	105	0.55	0.46	0.51	0.88	0.42	6.2	10.3	15.0
J	6031	98	0.81	0.60	0.31	1.05	0.45	6.3	6.7	11.6
K	5911	99	0.50	0.80	0.59	1.11	0.73	12.6	20.0	27.4
L	3374	57	0.44	0.32	0.29	0.62	0.46	4.9	7.7	13.9
M	4045	61	0.30	0.44	0.57	0.78	0.58	7.8	9.0	11.1
Mean $\pm$ std	3940	61	0.45 $\pm$ 0.19	0.42 $\pm$ 0.22	0.38 $\pm$ 0.14	0.75 $\pm$ 0.25	0.48 $\pm$ 0.10	7.1 $\pm$ 2.4	9.1 $\pm$ 3.9	14.7 $\pm$ 7.4
<i>Held-out flights (excluded from tuning)</i>										
N	2521	37	0.16	0.15	0.33	0.40	0.37	4.4	7.6	7.9
O	4679	62	0.44	0.60	0.76	1.06	0.62	10.2	16.2	12.0
P	6509	99	0.53	0.87	0.62	1.19	0.50	5.7	12.9	26.0
Held-out mean	4570	66	0.38	0.54	0.57	0.88	0.50	6.8	12.2	15.3

TinyCMax (Adapted Version)

The contrast-maximization experiments on flapping-wing event data use an adapted version of TinyCMax, provided in the `tinycmax/` subfolder of the project repository (<https://github.com/TijnVennink/onboard-evio-fw-mav>). It is adapted from the original TinyCMax implementation of [25], available at <https://github.com/Huizerd/tinycmax>, with modifications to:

- Input formatting for GenX320 event streams,
- Temporal window handling and accumulation sweeping