Real-Time Video Stabilization for Embedded Systems

By: Aditya Pola

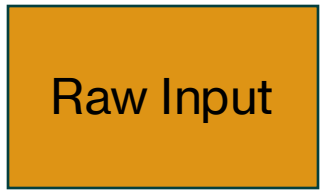
Motivation

Handheld cameras introduce unintentional jitter from small hand movements

This motion reduces visual quality and affects computer vision tasks

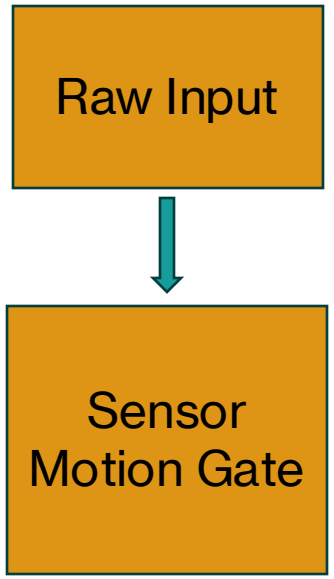
Goal: estimate camera motion and remove unintended movement

Pipeline Overview



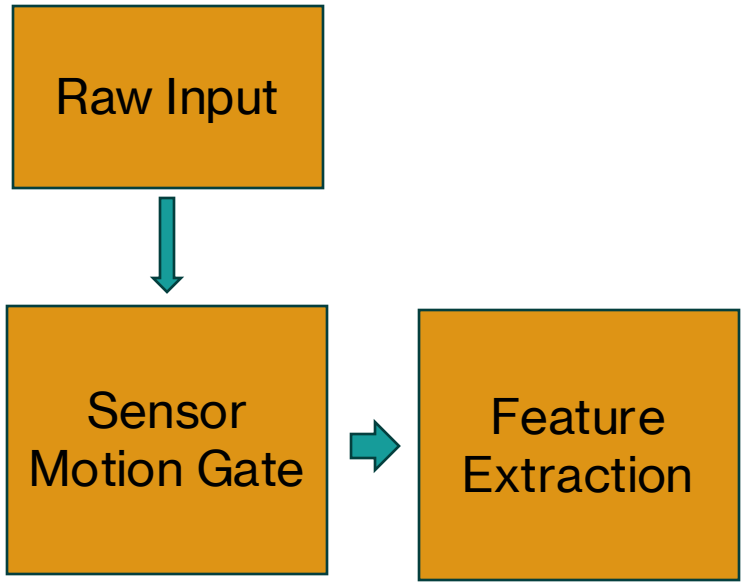
Obtain live video input from camera

Pipeline Overview



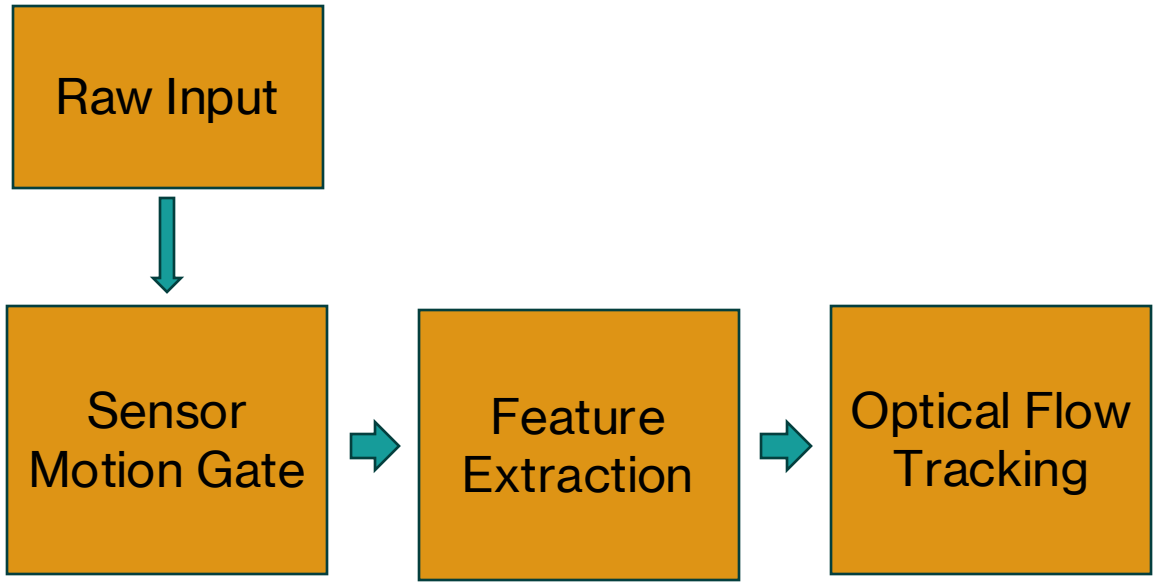
Utilize gyroscope to identify erratic motions that violate tracking assumptions and shutoff processing pipeline

Pipeline Overview



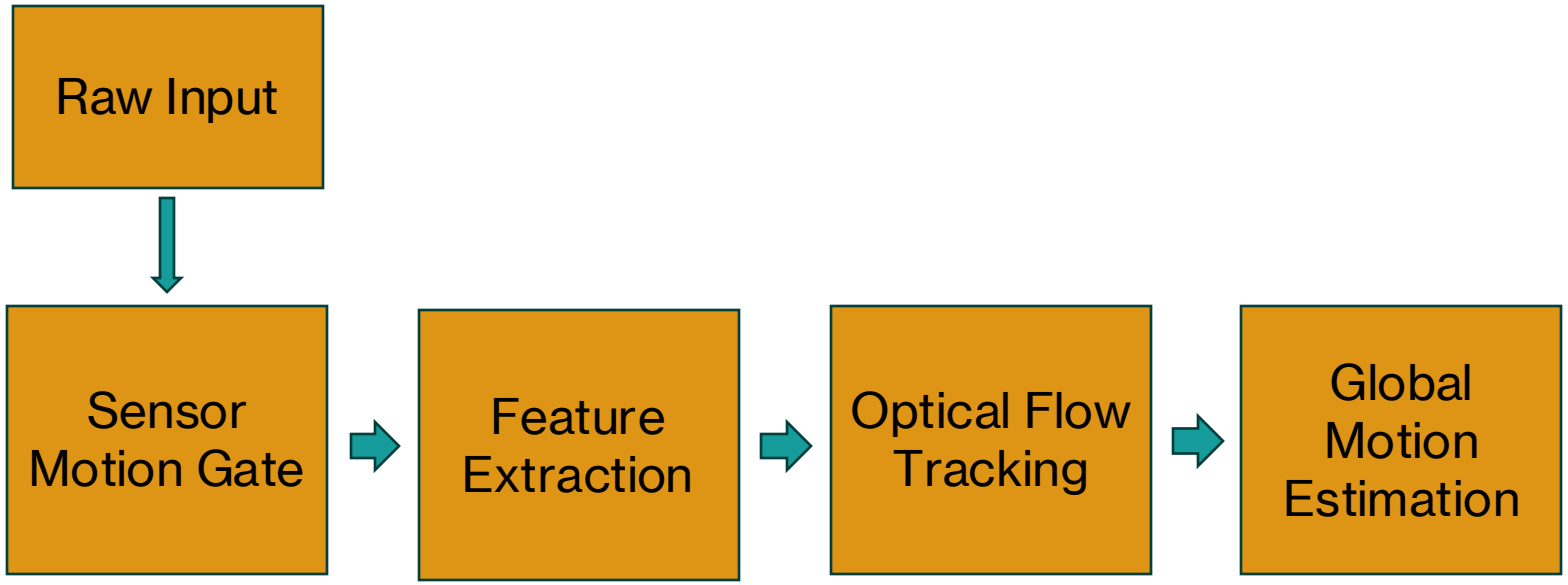
Identify stable points in the image that can be reliably tracked between frames.

Pipeline Overview



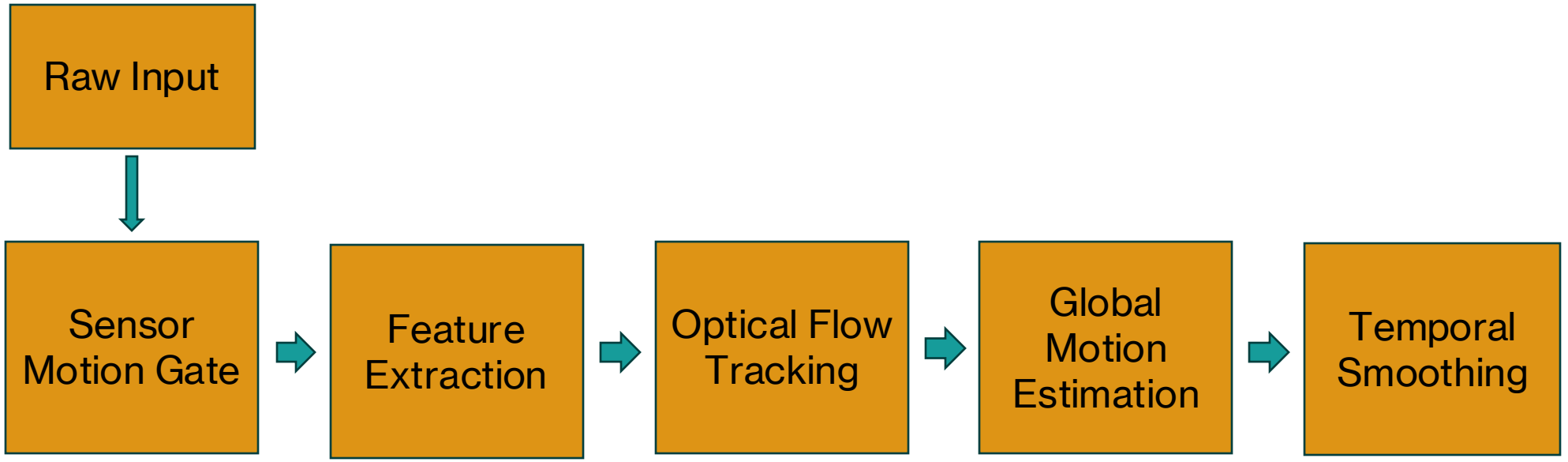
Track how detected feature points move between consecutive frames.

Pipeline Overview



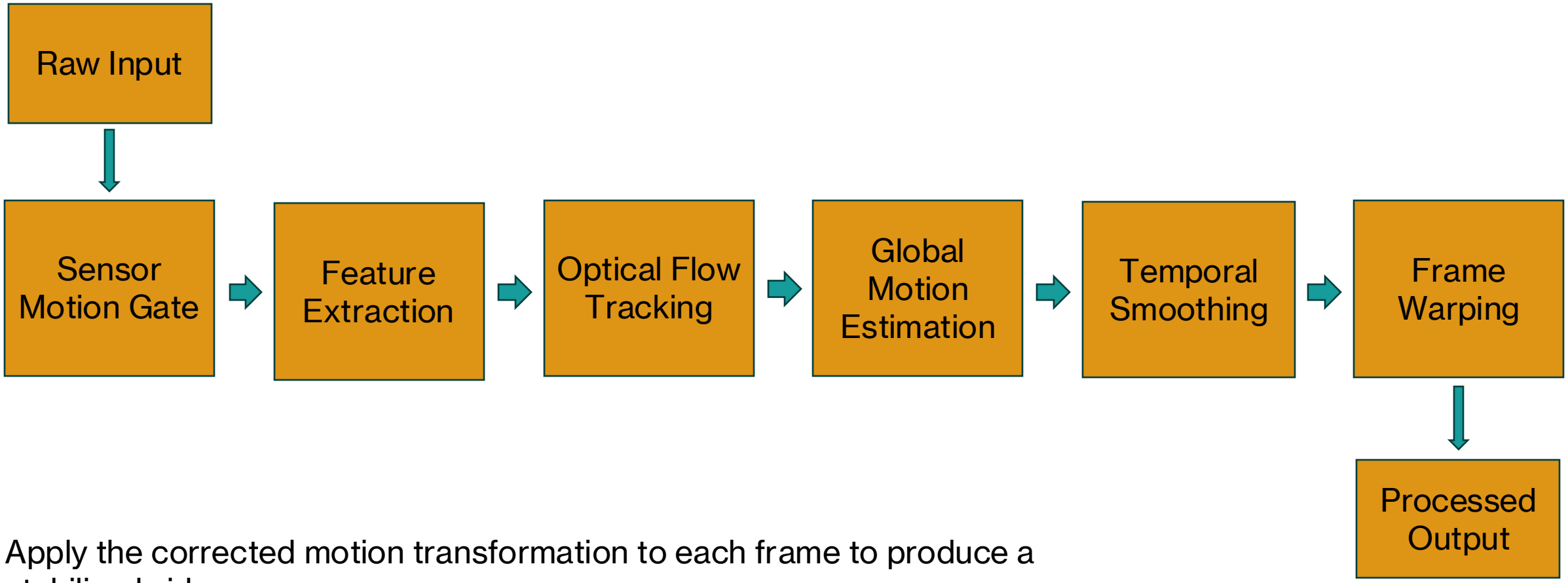
Combine feature motions to estimate the overall camera movement between frames.

Pipeline Overview



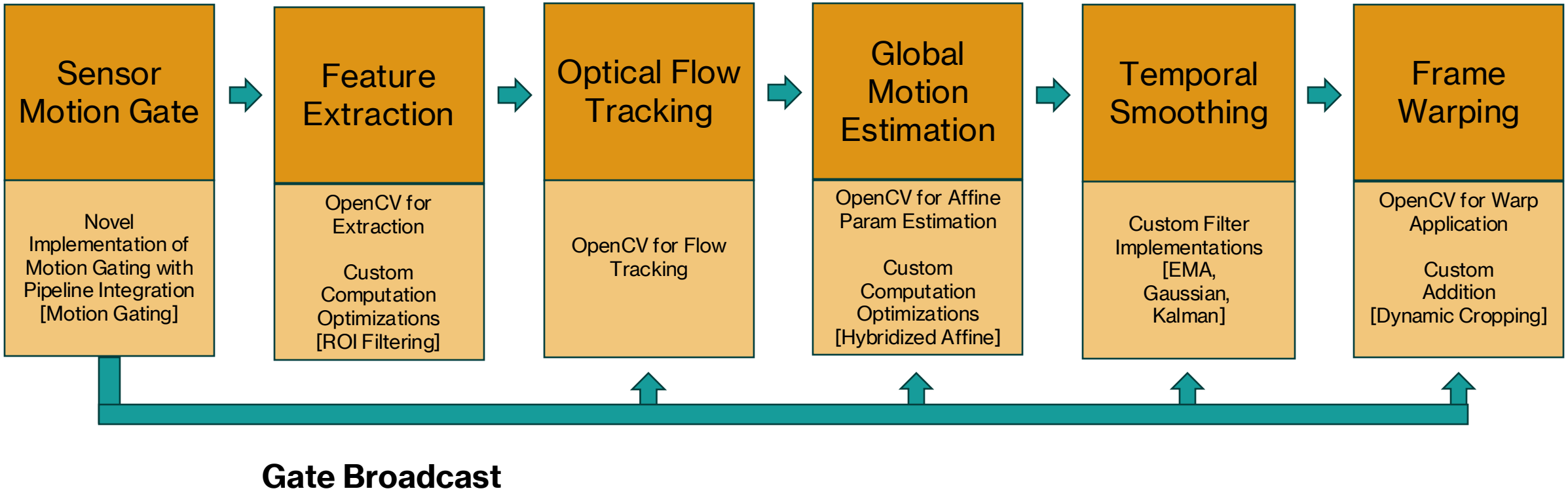
Smooth the estimated camera motion over time to remove high-frequency jitter.

Pipeline Overview



Apply the corrected motion transformation to each frame to produce a stabilized video.

My Contributions



Sensor Motion Correction [Gyro Gate Theory]

- Gyroscopes measure the **angular velocity** of the camera at each frame
- Lucas-Kanade assumes **brightness constancy** — rapid motion causes large pixel displacements that violate this assumption
- Corrupted optical flow vectors produce motion estimates **worse than no estimate at all**

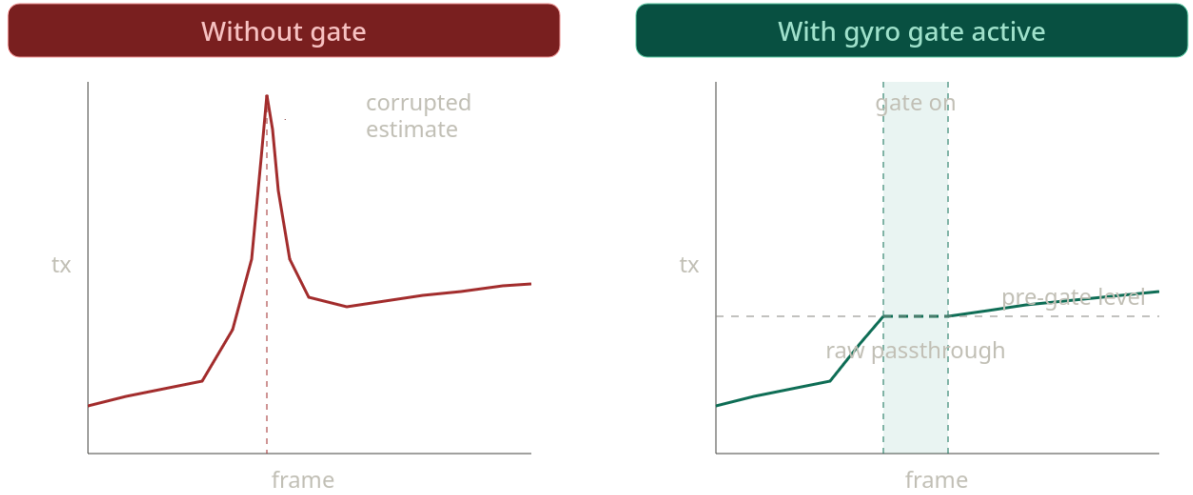
$$\|\omega(t)\| = \sqrt{g_x^2 + g_y^2 + g_z^2}$$

- If magnitude exceeds threshold, the **gate activates** and the frame is passed through unchanged:

$$\|\omega(t)\| > \tau_\omega$$

Sensor Motion Correction [Gate Broadcast]

- Gate evaluated **once per frame** before any visual processing begins
- When active, all downstream stages are bypassed from Feature Extraction through Frame Warping
- Operates **independently of scene content**, effective in any lighting and low-texture environments
- Feature re-detection triggered on gate deactivation to ensure **clean pipeline restart**



Feature Extraction [Shi-Tomasi Corners]

- Good tracking points must contain **intensity variation in multiple directions**
- Local image structure is captured by the **structure tensor**

$$M = \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

- I_x - Horizontal image gradient $\quad \mathrm{I}_x = \frac{\partial I}{\partial x}$
- I_y - Vertical image gradient $\quad \mathrm{I}_y = \frac{\partial I}{\partial y}$

- Eigenvalues λ_1 & λ_2 measure **directional intensity variation in orthogonal directions**

Feature Extraction [Shi-Tomasi Corners]

- Corners are selected using the **minimum eigenvalue test**

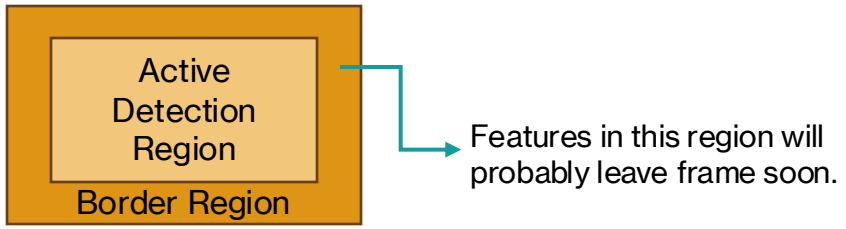
$$\min(\lambda_1, \lambda_2) > \tau$$

- Candidates are selected if:
  - Both eigenvalues are large -> Strong intensity changes in both directions

- Selected features are:
  - **Stable under small motion**
  - **Easy to localize across frames**

Feature Extraction [ROI Border Exclusion]

- Border features are likely to **leave the frame** between consecutive steps
- Tracking them wastes compute budget on points that **cannot be maintained**



$$\text{margin} = 0.10 \times \text{frame dimension}$$

- Features detected **only within the inner region**
- Reduces wasted Lucas-Kanade iterations on untrackable points
- Improves tracking efficiency at **no quality cost [1]**

Optical Flow Tracking [Lucas-Kanade Method]

- After detecting corners, we **track their motion between frames**
- Lucas–Kanade assumes **brightness constancy**:

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

- Pixel intensity remains approximately **constant between subsequent frames**
- Linearizing the motion gives the **optical flow constraint equation**

$$I(x+u, y+v, t+1) \approx I(x, y, t) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = I_x u + I_y v + I_t = 0$$

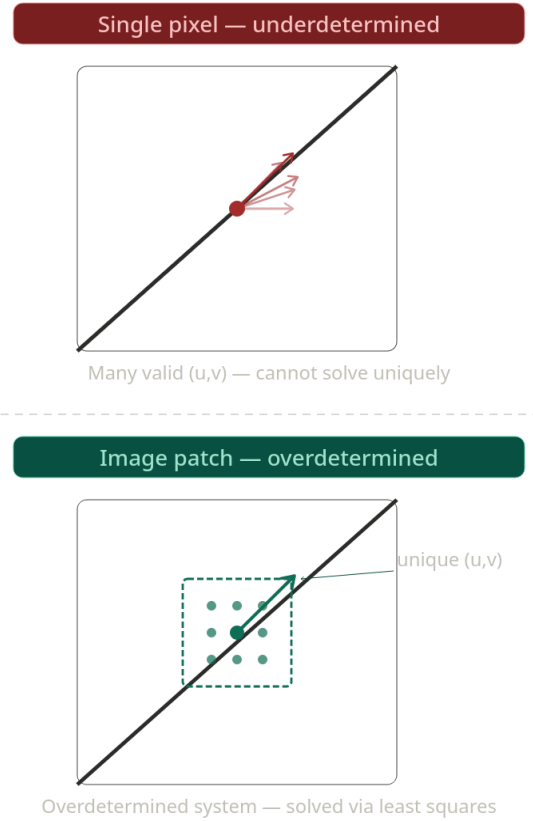
- (u,v) - pixel displacement between frames

Optical Flow Tracking [Lucas-Kanade Method]

- A single pixel equation cannot determine both motion components (u,v)
- Lucas–Kanade assumes **neighboring pixels move together**
- Motion is estimated over a **small image patch**
- Solve the least-squares system

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix}$$

- Produces **feature correspondences between frames**
- These correspondences are used to estimate **global camera motion**



Global Motion Estimation [Affine Transform]

- Optical flow produces **many individual feature displacements**
- Not all motion corresponds to **camera motion**
  - moving objects
  - tracking noise
  - mismatched features
- Goal: estimate the **dominant camera motion** between frames
- We model camera motion using an **affine transformation**

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Global Motion Estimation [Affine Transform]

- The affine transform models several geometric functions to estimate motion

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Translation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

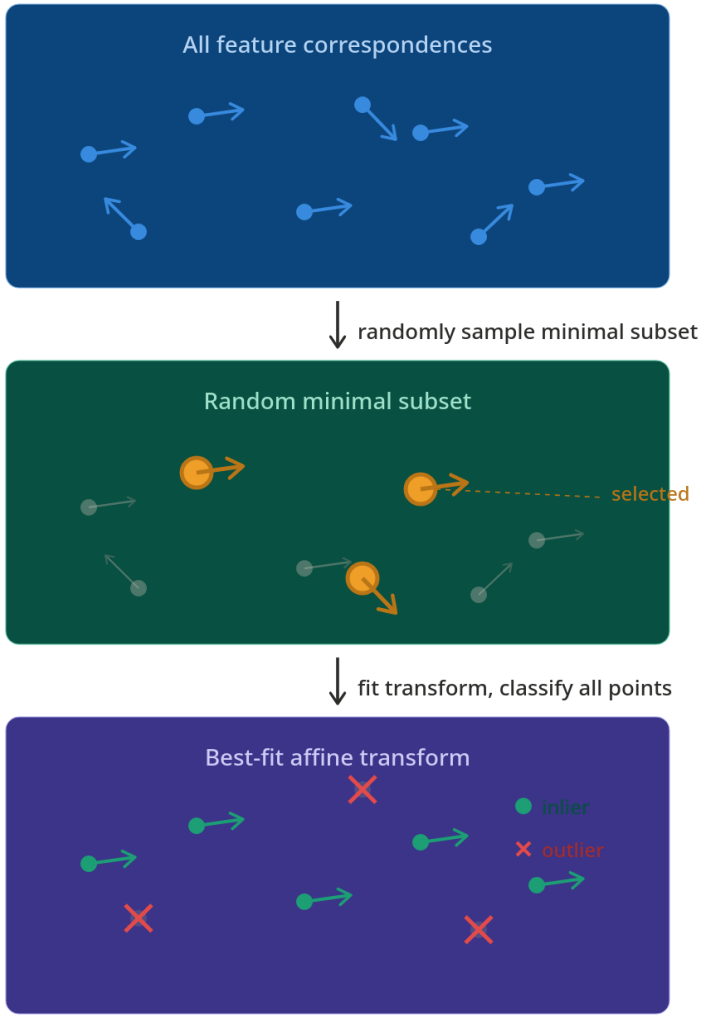
Scale

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Shear

Global Motion Estimation [RANSAC]

- Some feature matches correspond to **independent object motion**
- Direct fitting would be **sensitive to outliers**
- RANSAC estimates the affine transform **robustly**
- **Procedure**
  - Randomly select a **small subset of feature matches**
  - Compute candidate **affine transform**
  - Apply transform to all matches
  - Identify **inliers** (points consistent with motion)
  - Repeat for finite interval and choose model with **largest inlier set**



Global Motion Estimation [Hybridized Affine]

- RANSAC returns a **similarity transform**, encoding rotation plus uniform scale
- Scale extracted from returned affine coefficients:

$$s = \sqrt{a^2 + b^2}$$

- For typical handheld motion, true inter-frame scale change is **negligible [1]**
- Unconstrained scale causes **slow cumulative drift** over hundreds of frames
- If $|s-1|<\epsilon$, transform reclassified as **rigid**, pure rotation at unit scale:

$$\theta = \arctan_2(b,\, a) \quad \Rightarrow \quad \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

- Translation components **preserved**

Temporal Smoothing [Theory]

- Frame-to-frame affine estimation produces a **sequence of motion parameters**
- Raw estimates contain **high-frequency jitter** and directly applying them produces unstable video
- Goal: smooth the motion trajectory while **preserving intentional movement**
- Parameters accumulated into **cumulative path**:

$$\mathrm{R}_k = R_{k-1} \cdot A_k$$

- Four scalar parameters extracted and filtered independently: **translation** (tx,ty) **rotation** θ, **log-scale** ln(s)
- Three filter designs implemented and **selectable at runtime**: EMA, Gaussian, Kalman

Temporal Smoothing [UMA]

- Simplest approach, a **uniform average** over sliding window of N most recent frames

$$\bar{p}_k = \frac{1}{N} \sum_{i=k-N+1}^{k} p_i$$

- All frames weighted **equally regardless of recency**
- Easy to implement with no assumptions about motion process
- Limitation: **poor frequency response**, slow to attenuate jitter and introduces lag on genuine motion changes
- Limitation: sparse window at startup causes **unreliable estimates during warmup**

Temporal Smoothing [EMA]

- Weights each new measurement by α and prior estimate by (1−α), giving **recent frames exponentially more influence**

$$\bar{p}_k = \alpha \cdot z_k + (1 - \alpha) \cdot \bar{p}_{k-1}$$

- $\alpha \in (0,1]$ — near zero: heavy smoothing; near one: follows raw trajectory
- Implementation uses $\alpha = 0.3$ (moderate smoothing)
- State **initialized to first measurement**, avoiding zero-initialization transient
- Limitation: fixed α cannot simultaneously react to fast genuine motion and reject noise

Temporal Smoothing [Gaussian]

- Convolves full parameter history with a **Gaussian kernel** over window W

$$g_w = \frac{1}{Z} \exp\left(-\frac{w^2}{2\sigma^2}\right), \quad w \in \left[-\frac{W-1}{2}, \frac{W-1}{2}\right]$$

$$\bar{p}_k = \sum_w g_w \cdot p_{k+w}$$

- **Bell-shaped profile** weights recent frames more strongly than distant ones
- σ controls the tradeoff between **smoothing strength** and **responsiveness**
- Improvement over EMA: no stale-frame bias and a **cleaner frequency cutoff**
- Rotation parameter θ **unwrapped** before smoothing to prevent 2π discontinuities

Temporal Smoothing [Kalman]

- Models each motion parameter as a **hidden state evolving over time**
- Prediction step:

$$\hat{x}_k^- = x_{k-1}$$

- Correction step:

$$x_k = \hat{x}_k^- + K\left(z_k - \hat{x}_k^-\right)$$

- K∈[0,1], the **Kalman gain**, determines trust in new measurement vs. prior estimate
- Unlike EMA and Gaussian, explicitly **minimizes measurement residual** at each step
- State **initialized to first measurement**, avoiding the EMA warmup transient

Frame Warping [Corrective Warp]

- Rk is the **cumulative path**, representing total camera displacement from frame 0 to k

$$R_k = R_{k-1} \cdot A_k$$

- Smoother produces **smoothed target path** R¯k from filtered parameters
- R¯k represents where the camera **should have been** at frame k
- Rk−1 **undoes the actual path** and R¯k **re-applies the smooth version**

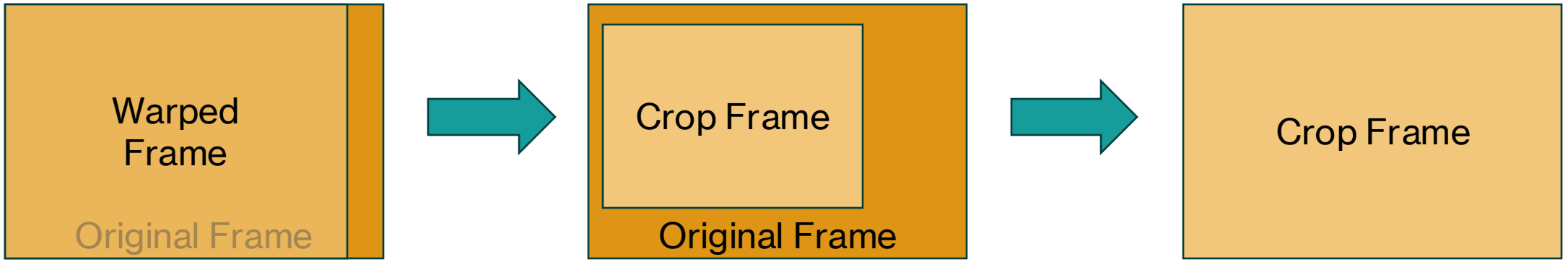
$$H = \bar{R}_k \cdot R_k^{-1}$$

- Transformation can then be applied to input frame to stabilize

$$\mathbf{x}_{\text{stab}} = H\,\mathbf{x}$$

Frame Warping [Dynamic Crop]

- Corrective warp shifts frame contents, leaving **unfilled border regions**
- A fixed margin wastes field of view when the camera is steady



- Crop rectangle computed **dynamically** from full warp history each frame
- Four frame corners **back-projected** through each stored inverse warp
- **Largest axis-aligned rectangle** covered by all warps is selected

Pseudocode

```
function STABILIZE(frame, prevGray, prevPts, pathR, filter):
    gray = toGrayscale(frame)
    if prevGray is empty:
        prevGray = gray;
        prevPts = detectShiTomasiCorners(gray)
        return frame                          // skip first frame
    gyroNorm = readGyroscopeMagnitude()
    if motionGatingEnabled and gyroNorm > GATE_THRESHOLD:
        return frame                          // skip during sharp motion
    currPts, status = lucasKanade(prevGray, gray, prevPts)
    goodPrev, goodCurr = filterByStatus(prevPts, currPts, status)
    if |goodPrev| >= 4:
        A, inliers = estimateAffineRANSAC(goodPrev, goodCurr)
        A = applyHybridModel(A)               // constrain to rigid if scale ~ 1
    else:
        A = identity
    pathR = pathR * lift3x3(A)               // accumulate cumulative path
    Q     = filter.smooth(pathR)             // smoothed target path
    warp  = Q * inverse(pathR)               // corrective transform
    stabilized = warpAffine(frame, warp)
    cropRect   = computeCropFromWarpHistory()
    output     = resize(crop(stabilized, cropRect))
    prevGray = gray
    prevPts  = goodCurr  (re-detect if |goodCurr| < MIN_FEATURES)
```

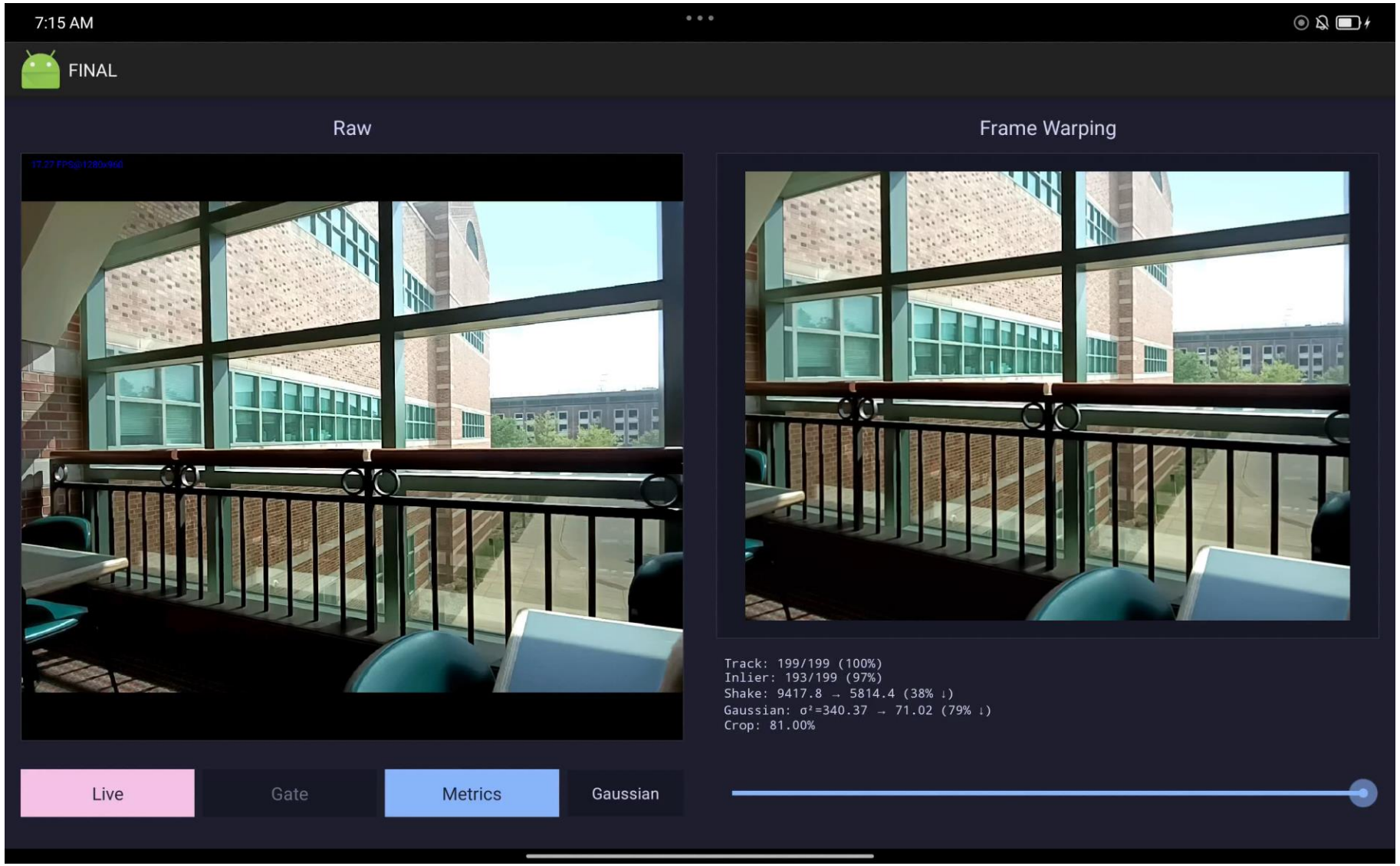
Software Hierarchy

File	Responsibility
`MainActivity.java`	Activity entry point. Manages UI (dual feed, stage slider, filter spinner, gate/metrics buttons), hosts the camera frame callback, and drives the per-frame pipeline.
`PipelineUtils.java`	Stateless utility class holding tunable constants and shared helpers: affine/homogeneous conversion, Gaussian smoothing, angle unwrapping, similarity parameter extraction, warp and crop computation, and visualization routines.
`KalmanPathFilter.java`	Kalman smoother over $(t_x, t_y, \theta, \ln s)$ with angle-unwrapping and warp history for dynamic crop.
`GaussianPathBuffer.java`	Gaussian smoother. Accumulates parameter history and re-smooths via Gaussian convolution each frame.
`ExponentialMovingPathBuffer.java`	EMA smoother. Applies $\bar{p}_k = \alpha z_k + (1-\alpha)\bar{p}_{k-1}$ with $\alpha = 0.3$ to each of the four path parameters independently.
`UniformMovingPathBuffer.java`	UMA smoother. Applies a uniform average over a sliding window of N recent frames to each path parameter.
`GyroReader.java`	Wraps `SensorManager` to expose gyroscope readings (g_x, g_y, g_z) and a gate-ready flag; registers/unregisters with the activity lifecycle.
`MetricsCollector.java`	Collects per-frame statistics and emits tagged CSV lines to ADB logcat for offline analysis.

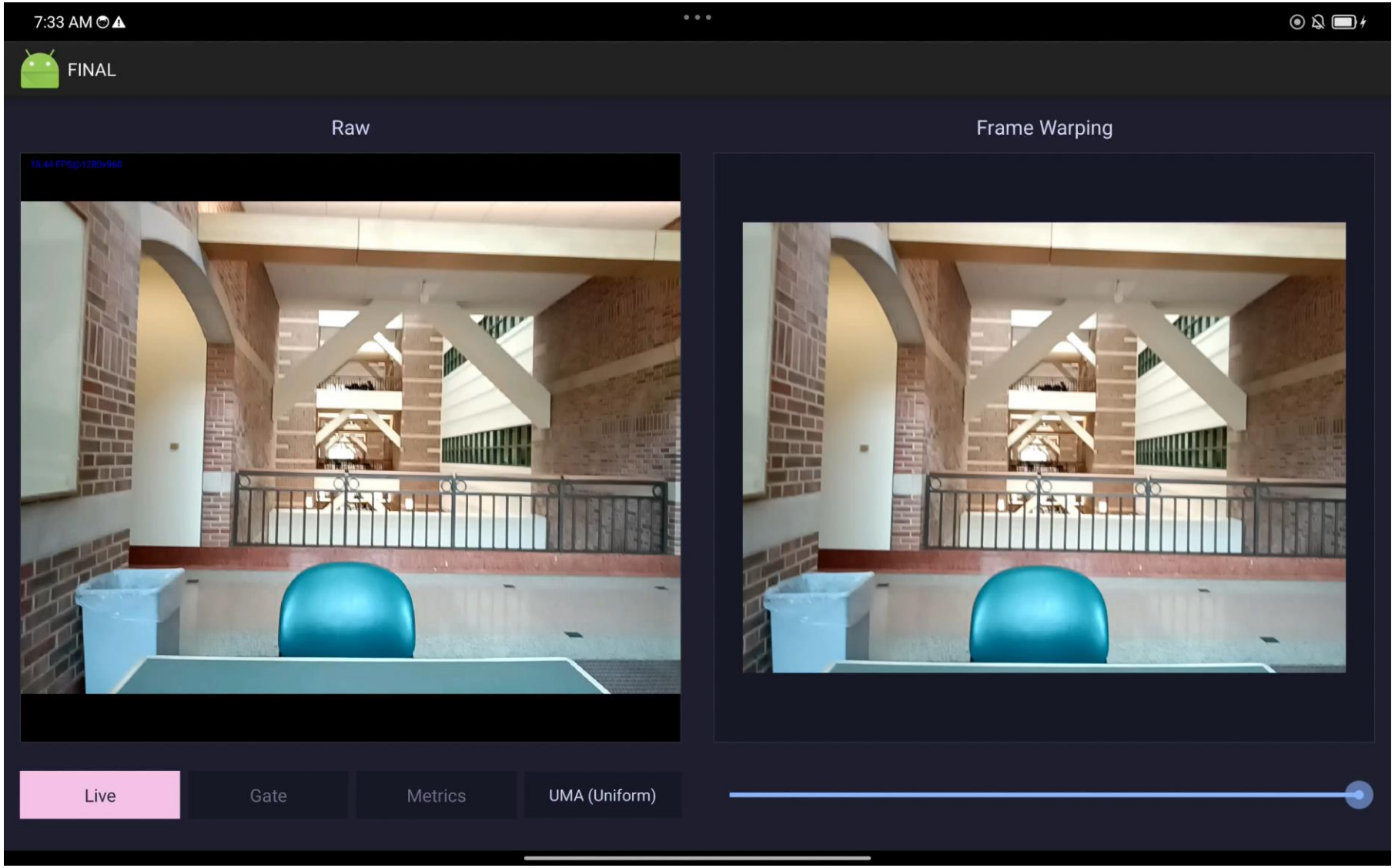
Script	Purpose
`capture_metrics.sh`	Filters ADB logcat for `METRICS_CSV` lines and writes them to a timestamped CSV. Used for all four filter evaluation runs.
`debug_crashes.sh`	Monitors logcat for fatal exceptions and ANRs in `com.ece420`, writing crash logs to disk for development debugging.

Demo

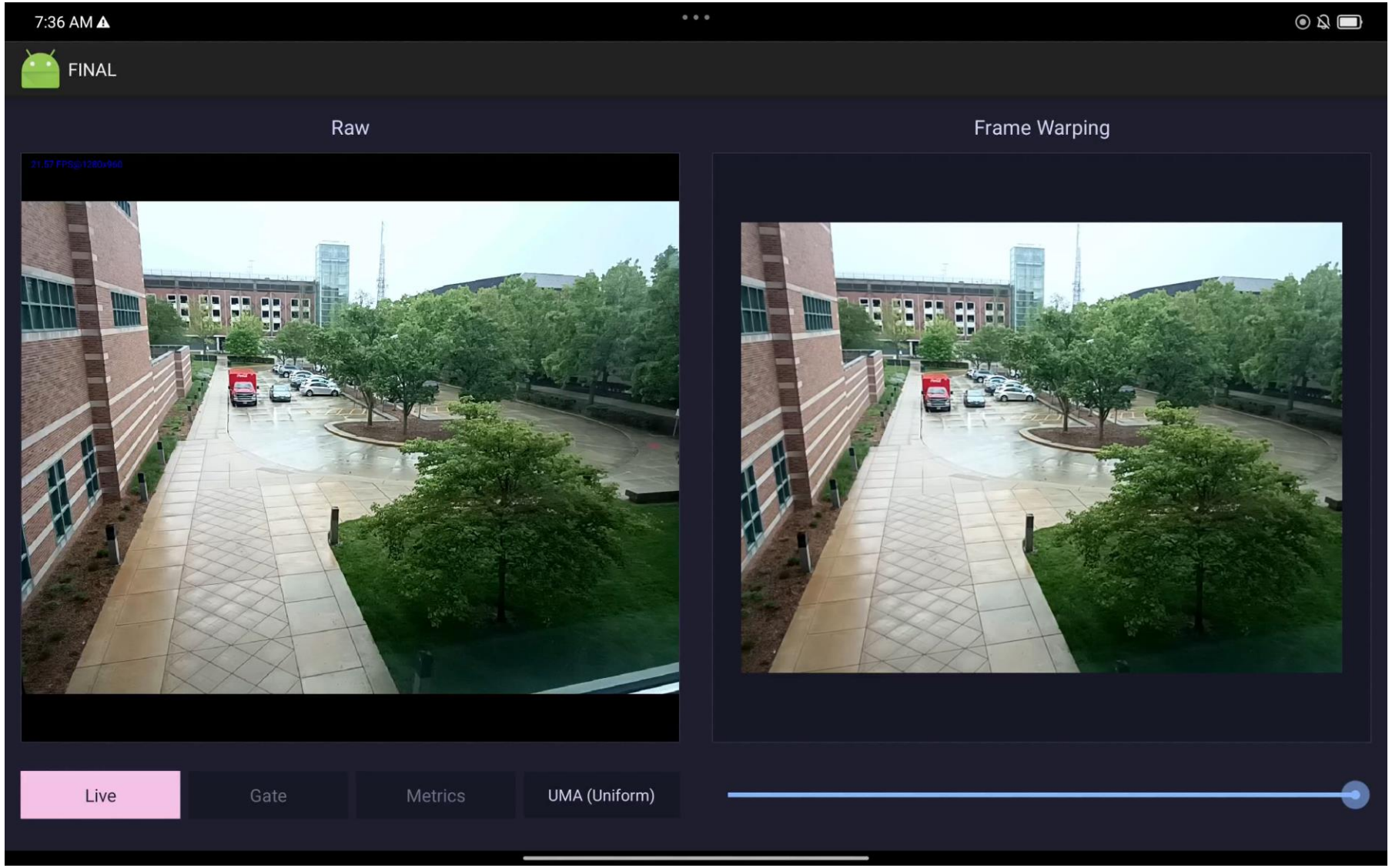
Results [Demo]



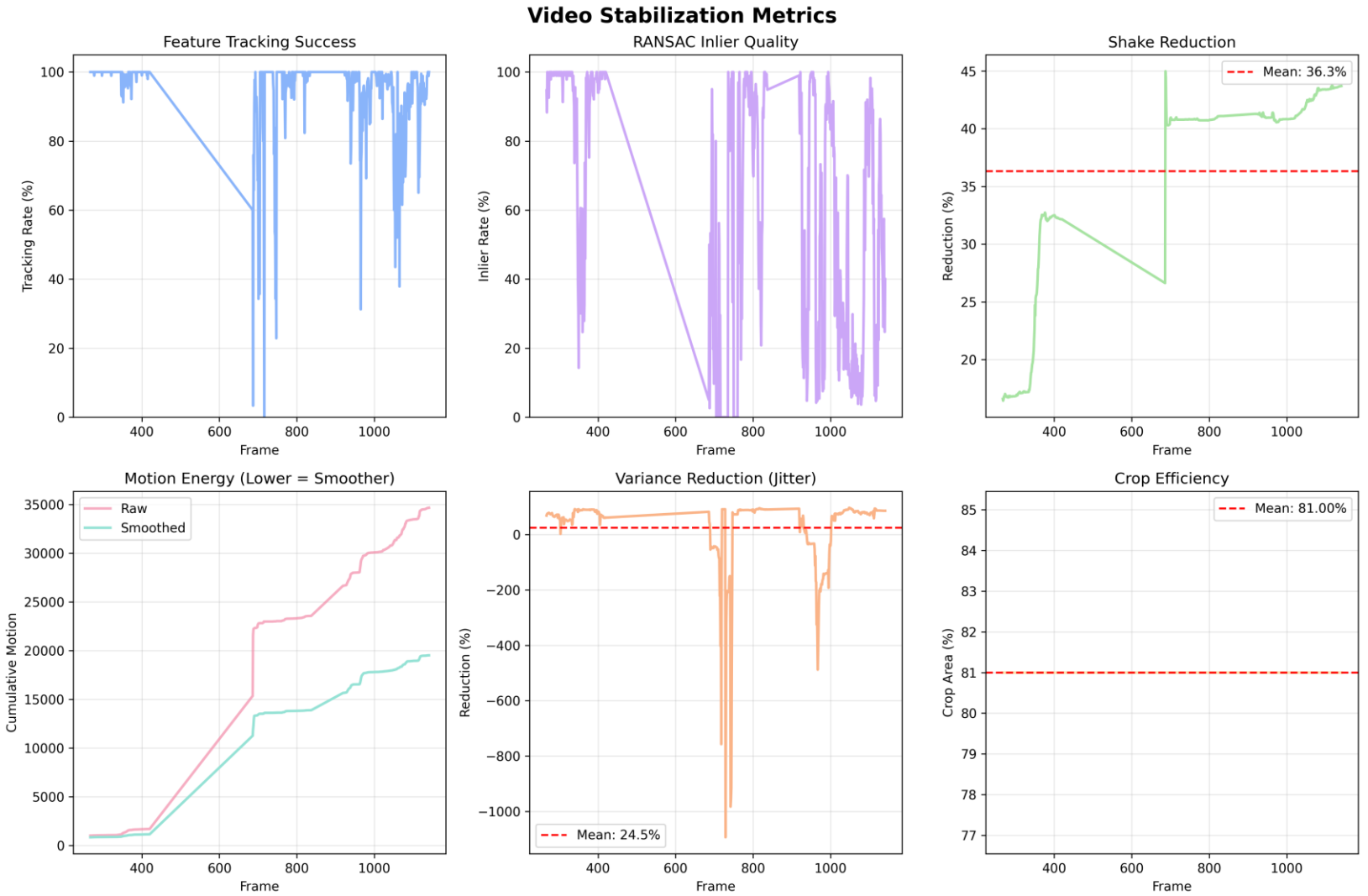
Results [Demo]



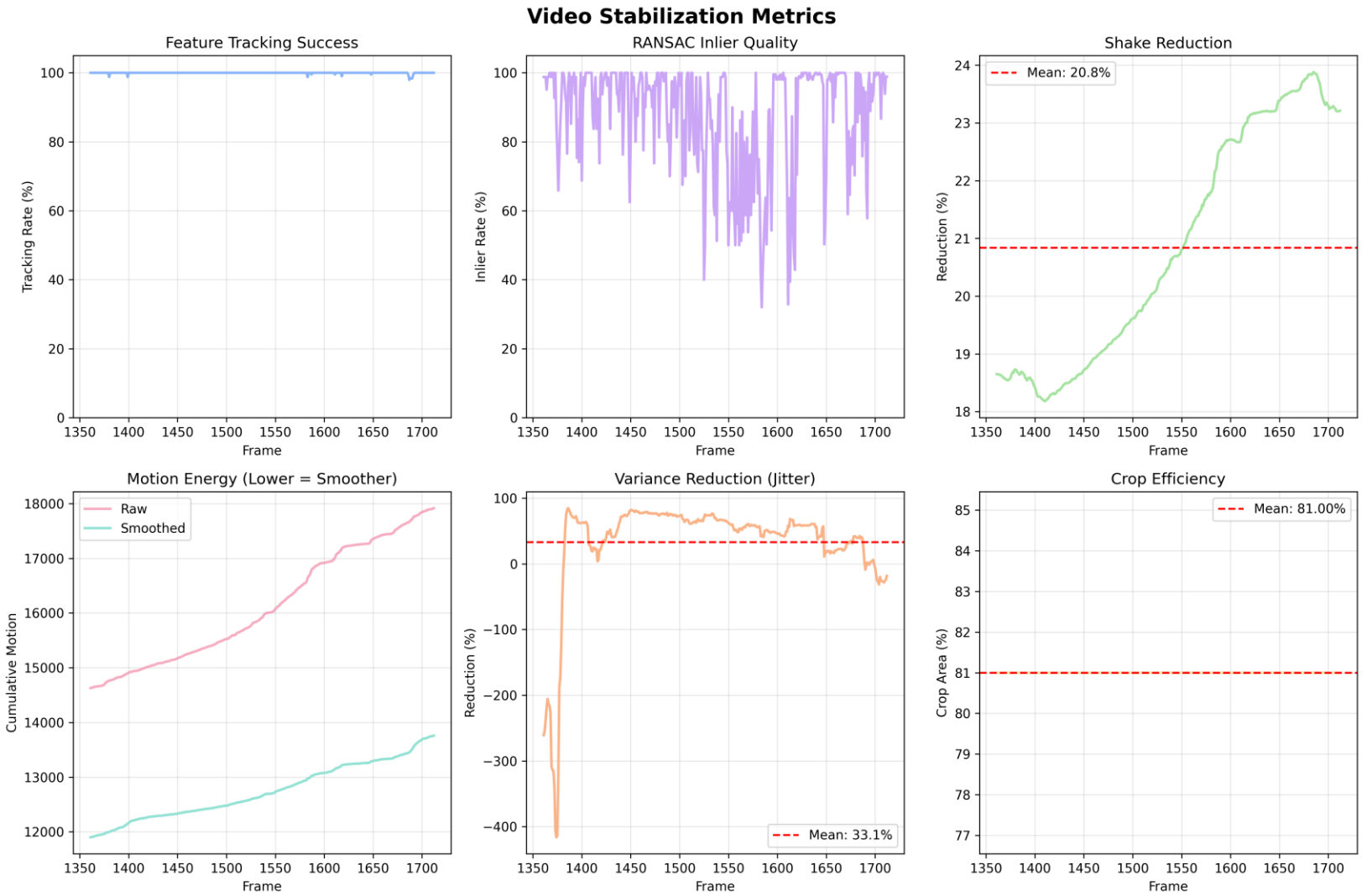
Results [Demo]



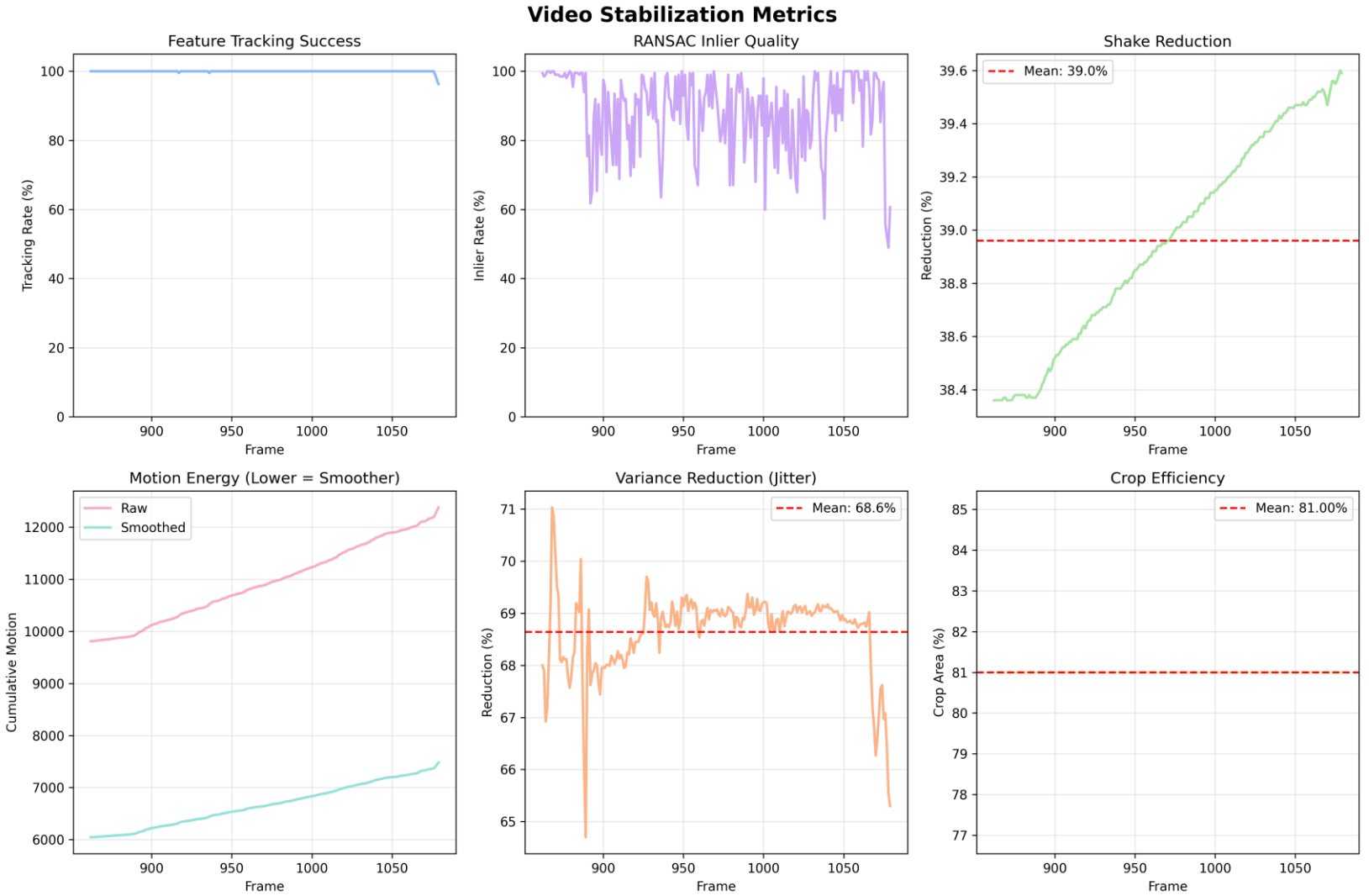
Filter Evaluation [UMA]



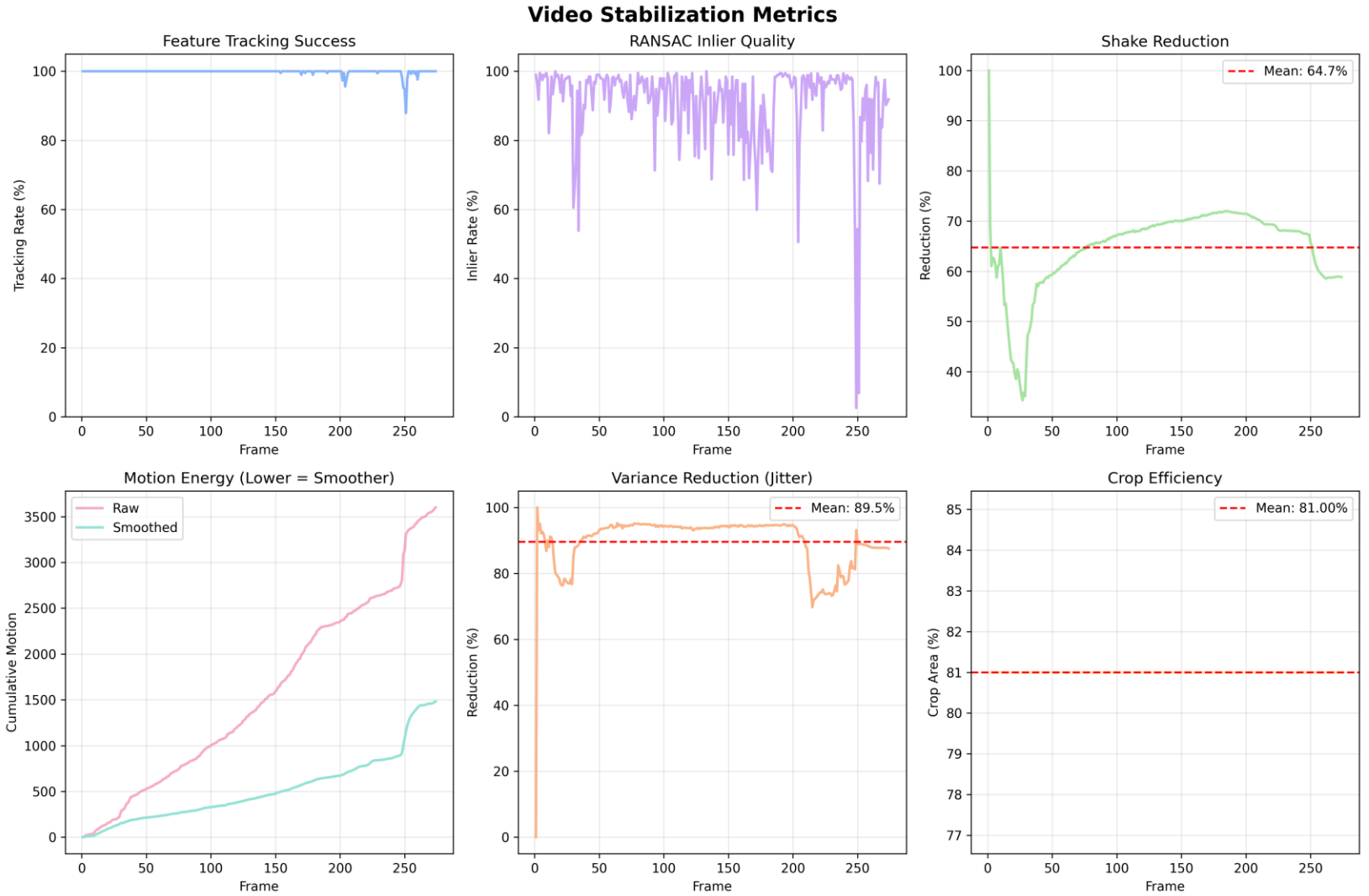
Filter Evaluation [EMA]



Filter Evaluation [Gaussian]



Filter Evaluation [Kalman]



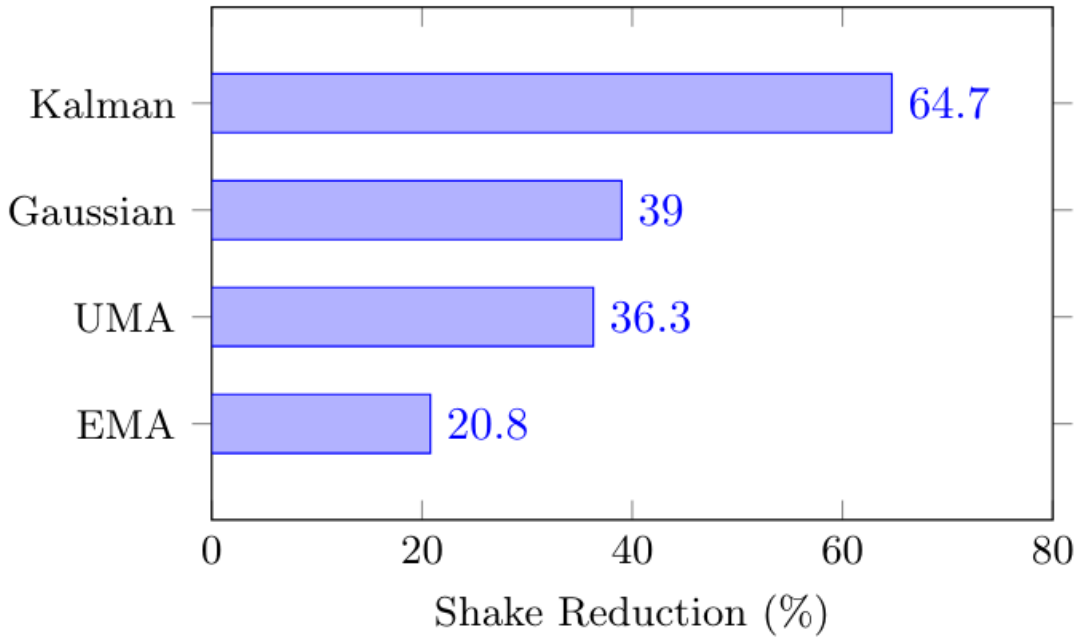
Results [Summary]

Filter	Track Rate	Inlier Rate	Shake Reduction	Var. Reduction	Crop
Kalman	99.84%	90.79%	**64.7%**	**89.9%**	81.0%
Gaussian	99.96%	88.31%	39.0%	68.6%	81.0%
EMA	99.96%	87.50%	20.8%	33.9%	81.0%
UMA	94.08%	61.23%	36.3%	24.5%	81.0%

- Feature tracking exceeded **99.8%** across all three configurations, confirming reliable upstream pipeline operation
- Crop efficiency **identical at 81%** across all filters, confirming stabilization quality is decoupled from field-of-view cost
- Kalman achieves **64.7% shake reduction** and **89.9% variance reduction**, the strongest result of the three

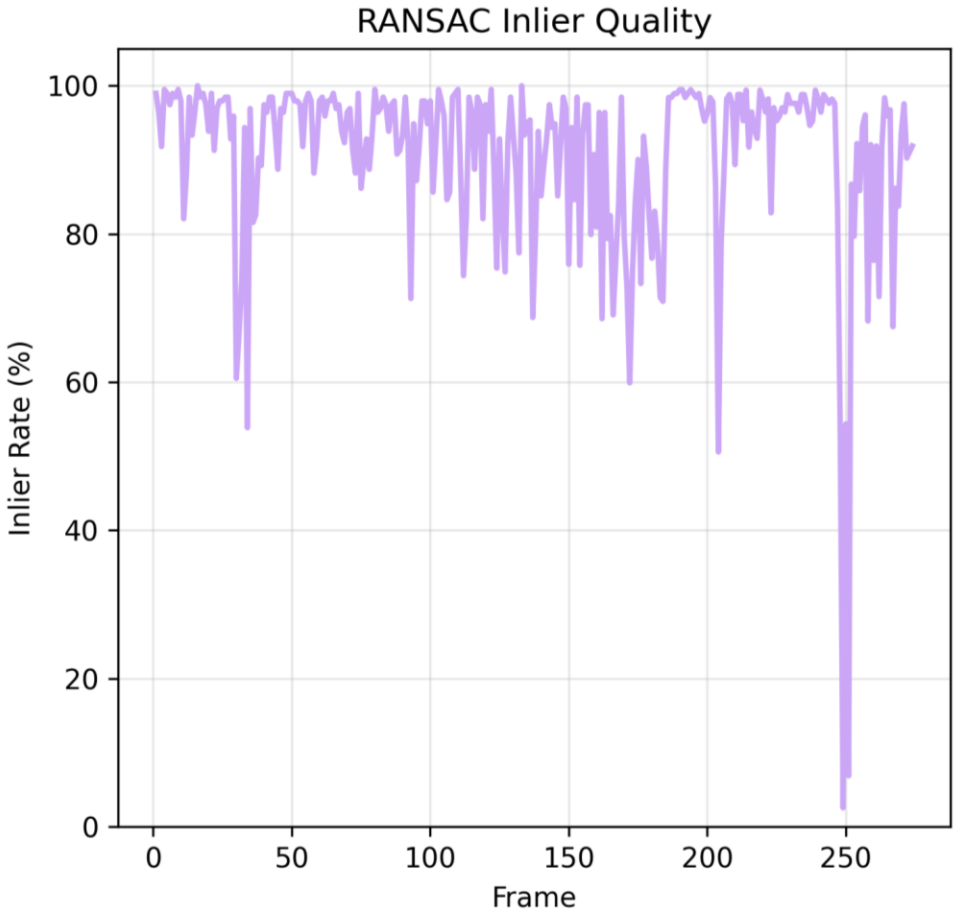
Results [Analysis – Filter Comparison]

- EMA's **fixed decay factor** cannot simultaneously react to genuine motion and reject noise
- Gaussian's **bell-shaped kernel** outperforms EMA by weighting recent frames more cleanly
- Kalman's **state-space residual minimization** produces the strongest result by a wide margin



Results [Analysis – Motion Gating]

- Transient inlier drops to **2.6%** confirm the exact failure mode the gate protects against
- Frames below τ_ω but still fast enough to corrupt flow represent the **residual unhandled gap**
- Motivates replacing binary threshold with **joint gyroscope + inlier gating** as an extension



Conclusions

- Real-time stabilization successfully demonstrated on Android tablet at **interactive frame rates**
- Gyro-based motion gating and **four-filter design progression** validated quantitatively as original contributions
- **Kalman filter** achieves strongest result: 64.7% shake reduction, 89.9% variance reduction

Filter	Track Rate	Inlier Rate	Shake Reduction	Var. Reduction	Crop
Kalman	99.84%	90.79%	**64.7%**	**89.9%**	81.0%
Gaussian	99.96%	88.31%	39.0%	68.6%	81.0%
EMA	99.96%	87.50%	20.8%	33.9%	81.0%
UMA	94.08%	61.23%	36.3%	24.5%	81.0%

Limitations

- **Moving objects and multiple subjects** — large foreground motion biases the RANSAC affine estimate when moving objects dominate the frame
- **Low-texture scenes** — few corners degrade feature detection and increase re-detection frequency, introducing brief path discontinuities
- **Fixed gate threshold** — moderate-speed motion below τ_ω can still corrupt optical flow, and the threshold cannot adapt to varying scene conditions
- **EMA transient warmup** — zero initialization causes brief overcorrection before filter convergence
- **Parallax and depth variation** — affine model assumes roughly planar scene geometry and produces distortion in scenes with strong depth changes

Extensions

Multithreaded Pipeline

- Distribute motion estimation, smoothing, and warping **across threads** (producer-consumer model)
- Sweep design space to **optimize power/performance** tradeoffs for embedded deployment

Adaptive Motion Gating

- Gate on joint **gyroscope + RANSAC** inlier signal rather than angular velocity alone
- High inlier ratio can override the gate; contribution weighted by inlier ratio within non-gated frames

Extended Motion Model & Improved Testing Environment

- Standardized mechanical shake source for controlled, repeatable filter evaluation
- Full projective homography to handle perspective distortion from large rotations

References

- Lim, Anli & Ramesh, Bharath & Yang, Yue & Xiang, Cheng & Gao, Zhi & Lin, Feng. (2019). Real-Time Optical flow-based Video Stabilization for Unmanned Aerial Vehicles. Journal of Real-Time Image Processing. 16. 10.1007/s11554-017-0699-y.
- Deniz, O.; Bueno, G.; Bermejo, E.; Sukthankar, R. (2011). Fast and accurate global motion compensation. Pattern Recognition. Volume 44, Issue 12. Pages 2887–2901. ISSN 0031-3203. https://doi.org/10.1016/j.patcog.2010.10.019.
- https://zhangleibit.github.io/download/stabfr/dataset.html