Motion Detection and Object Tracking from Video: Implementation Comparison

Computational Project Analysis

November 25, 2025

1 Introduction

This document describes two complementary implementations of motion detection and object tracking systems designed to extract kinematic parameters from video sequences. The computational project aims to:

1. Extract the number of moving objects from video frames
2. Calculate motion derivatives: speed, acceleration, jerk, and jounce (in pixel units)
3. Apply clustering algorithms with various norm metrics
4. Implement thresholding and smoothing techniques
5. Convert results to real physical units where applicable
6. Analyze two scenarios: single object and multiple object tracking

Two implementations are provided:

- **From Scratch**: Pure NumPy implementation with DBSCAN clustering
- **With Libraries**: OpenCV-based implementation using contour detection

2 Mathematical Foundations

2.1 Kinematic Derivatives

Given a sequence of positions $\mathbf{p}_i = (x_i, y_i)$ sampled at frame rate f_{ps}, where the time step is $\Delta t = 1/f_{\text{ps}}$, we compute four orders of derivatives:

2.1.1 First Order: Velocity

The velocity vector at each frame is:

$$\mathbf{v}_i = \frac{\mathbf{p}_{i+1} - \mathbf{p}_i}{\Delta t} \tag{1}$$

The speed (magnitude of velocity) is:

$$\text{speed}_i = \|\mathbf{v}_i\| = \sqrt{v_{x,i}^2 + v_{y,i}^2} \tag{2}$$

Units: pixels per second (pix/s)

2.1.2 Second Order: Acceleration

The acceleration vector is the derivative of velocity:

$$\mathbf{a}_i = \frac{\mathbf{v}_{i+1} - \mathbf{v}_i}{\Delta t} \tag{3}$$

The acceleration magnitude is:

$$\text{acceleration}_i = \|\mathbf{a}_i\| = \sqrt{a_{x,i}^2 + a_{y,i}^2} \tag{4}$$

Units: pixels per second squared (pix/s^2)

2.1.3 Third Order: Jerk

Jerk is the derivative of acceleration:

$$\mathbf{j}_i = \frac{\mathbf{a}_{i+1} - \mathbf{a}_i}{\Delta t} \tag{5}$$

The jerk magnitude is:

$$\text{jerk}_i = \|\mathbf{j}_i\| = \sqrt{j_{x,i}^2 + j_{y,i}^2} \tag{6}$$

Units: pixels per second cubed (pix/s^3)

2.1.4 Fourth Order: Jounce

Jounce (also called snap) is the derivative of jerk:

$$\mathbf{s}_i = \frac{\mathbf{j}_{i+1} - \mathbf{j}_i}{\Delta t} \tag{7}$$

The jounce magnitude is:

$$\text{jounce}_i = \|\mathbf{s}_i\| = \sqrt{s_{x,i}^2 + s_{y,i}^2} \tag{8}$$

Units: pixels per second to the fourth power (pix/s^4)

2.2 Physical Unit Conversion

Converting from pixel units to real physical units requires a calibration parameter: the pixel scale γ (meters/pixel). This requires either:

1. A known reference object in the scene with known dimensions
2. Camera calibration data (focal length, sensor size)
3. Multi-camera triangulation

Given calibration factor γ [m/pix], the conversions are:

$$\text{speed}_{\text{phys}} = \text{speed}_{\text{pix}} \cdot \gamma \quad [\text{m/s}] \tag{9}$$

$$\text{acceleration}_{\text{phys}} = \text{acceleration}_{\text{pix}} \cdot \gamma \quad [\text{m/s}^2] \tag{10}$$

$$\text{jerk}_{\text{phys}} = \text{jerk}_{\text{pix}} \cdot \gamma \quad [\text{m/s}^3] \tag{11}$$

$$\text{jounce}_{\text{phys}} = \text{jounce}_{\text{pix}} \cdot \gamma \quad [\text{m/s}^4] \tag{12}$$

Without calibration information, physical unit conversion is **not possible**. This is a fundamental limitation when only video data is available without reference scale information.

3 Implementation 1: From Scratch (DBSCAN-Based)

3.1 Overview

The `motion_detection_from_scratch.ipynb` implements object detection and tracking using:

- Frame differencing for motion detection
- Custom DBSCAN clustering implementation
- Robust centroid tracking with patience mechanism
- Morphological operations (opening and dilation)

3.2 Algorithm Pipeline

3.2.1 1. Frame Preprocessing

For each input frame I_t:

$$I_t^{\text{gray}} = \text{BGR2GRAY}(I_t) \tag{13}$$

$$I_t^{\text{blurred}} = G_\sigma(I_t^{\text{gray}}) \tag{14}$$

where G_σ is a Gaussian blur with kernel size $(15, 15)$ and standard deviation σ.

3.2.2 2. Motion Detection via Frame Differencing

The motion mask is computed as:

$$D_t = |I_t^{\text{blurred}} - I_{t-1}^{\text{blurred}}| \tag{15}$$

$$M_t = \begin{cases} 255 & \text{if } D_t > \tau_{\text{motion}} \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

where $\tau_{\text{motion}} = 20$ is the motion threshold.

3.2.3 3. Morphological Operations

The motion mask undergoes morphological refinement:

Opening Operation:

$$M_t^{\text{open}} = \text{DILATE}(\text{ERODE}(M_t, K), K) \tag{17}$$

Dilation Operation:

$$M_t^{\text{final}} = \text{DILATE}(M_t^{\text{open}}, K, \text{iterations} = 2) \tag{18}$$

where K is a 3×3 or 5×5 structuring element.

Purpose:

- Opening removes small noise
- Dilation connects broken object parts and fills gaps

3.2.4 4. DBSCAN Clustering

The algorithm converts motion pixels to point coordinates and applies density-based clustering:

Distance Metric:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{19}$$

This is the standard **Euclidean norm** (L_2 norm).

Parameters:

- $\epsilon = 10$ pixels (neighborhood radius)
- min_samples $= 3$ (minimum points to form a cluster)
- Sampling limit: 5000 points (to handle large images)

3.2.5 5. Centroid Extraction

For each cluster C_k, the centroid is:

$$\mathbf{c}_k = \frac{1}{|C_k|} \sum_{p \in C_k} p \tag{20}$$

Only clusters with area ≥ 50 pixels are retained.

3.2.6 6. Robust Track Matching

A greedy nearest-neighbor matching strategy associates detections with existing tracks:

Parameters:

- $D_{\max} = 150$ pixels (maximum distance to match)
- $P_{\text{limit}} = 15$ frames (patience before removing track)

Algorithm 1 DBSCAN Clustering (Custom Implementation)

procedure DBSCAN($\mathcal{P}$, ϵ, min_samples)
    $N \leftarrow |\mathcal{P}|$
    labels $\leftarrow [-1, -1, \ldots, -1]$ ▷ Initialize all unvisited
    cluster_id $\leftarrow 0$
    for each point $p_i \in \mathcal{P}$ **do**
        if p_i is visited **then**
            continue
        end if
        Mark p_i as visited
        $N_\epsilon(p_i) \leftarrow$ RangeQuery($\mathcal{P}$, p_i, ϵ)
        if $|N_\epsilon(p_i)| <$ min_samples **then**
            label(p_i) $\leftarrow -1$ ▷ Noise point
        else
            label(p_i) $\leftarrow$ cluster_id
            seeds $\leftarrow N_\epsilon(p_i) \setminus \{p_i\}$
            while seeds not empty **do**
                $q \leftarrow$ pop(seeds)
                if q not visited **then**
                    Mark q as visited
                    $N_\epsilon(q) \leftarrow$ RangeQuery($\mathcal{P}$, q, ϵ)
                    if $|N_\epsilon(q)| \geq$ min_samples **then**
                        seeds $\leftarrow$ seeds $\cup N_\epsilon(q)$
                    end if
                end if
                if label(q) $= -1$ **then**
                    label(q) $\leftarrow$ cluster_id
                end if
                cluster_id $\leftarrow$ cluster_id $+ 1$

                return labels

Algorithm 2 Track Update with Patience Mechanism

procedure UPDATETRACKS(tracks, centroids)
  if no existing tracks **then**
    Initialize new track for each centroid
    return
  end if
  matched_new $\leftarrow \emptyset$
  matched_tracks $\leftarrow \emptyset$
  for each track t with last position $\mathbf{p}_t$ **do**
    (best_dist, best_idx) $\leftarrow (\infty, -1)$
    for each centroid $\mathbf{c}_j$ not yet matched **do**
      $d \leftarrow \|\mathbf{p}_t - \mathbf{c}_j\|$
      if $d <$ best_dist **then**
        (best_dist, best_idx) $\leftarrow (d, j)$
      end if
    end for
    if best_dist $< D_{\max}$ **then**
      Append $\mathbf{c}_{\text{best\_idx}}$ to track t
      patience(t) $\leftarrow P_{\text{limit}}$ ▷ Reset
      matched_new.add(best_idx)
      matched_tracks.add(t)
    end if
  end for
  for each unmatched track t **do**
    patience(t) $\leftarrow$ patience(t) $- 1$
  end for
  for each unmatched centroid $\mathbf{c}_j$ **do**
    Create new track with $\mathbf{c}_j$
  end for
end procedure=0

3.3 Implementation Details

3.3.1 Distance Metric in Clustering

The custom DBSCAN uses the **squared Euclidean distance** for efficiency:

$$d^2(p_i, p_j) = (x_i - x_j)^2 + (y_i - y_j)^2 \tag{21}$$

This is vectorized using NumPy's einsum operation:

```
dists2 = np.einsum('ij,ij->i', diffs, diffs)
neighbors = np.where(dists2 <= eps * eps)[0]
```

3.3.2 Sampling Strategy

For images with $N > 5000$ motion pixels:

1. Randomly sample 5000 points
2. Cluster the sample
3. Estimate cluster centroids from samples
4. For each cluster, find all full image points within 1.5ϵ of estimated centroid
5. Recompute accurate centroid using full point set

This balances computational efficiency with clustering accuracy.

3.4 Output

3.4.1 Video Output

Tracks are visualized in the output video:

- Red circles: Current object centroids
- Track IDs: Label next to each object
- Object count: Displayed in top-left corner

3.4.2 Derivative Analysis

For tracks with ≥ 30 frames, four plots are generated showing:

1. Speed vs. time (pixels per second)
2. Acceleration vs. time (pixels per second squared)
3. Jerk vs. time (pixels per second cubed)
4. Jounce vs. time (pixels per second to fourth power)

Each plot is saved as `track_[ID].png` in the `tracking_results/` folder.

4 Implementation 2: With Libraries (OpenCV-Based)

4.1 Overview

The `motion_detection_with_libraries.ipynb` provides a more practical implementation using OpenCV:

- Standard frame differencing
- Contour detection (faster alternative to DBSCAN)
- Same robust tracking mechanism
- Same derivative calculations

4.2 Algorithm Pipeline

4.2.1 1-3. Frame Preprocessing and Motion Detection

Identical to Implementation 1 (Equations 1-3).

4.2.2 4. Contour-Based Object Detection

Instead of clustering pixels, contours are extracted:

$$\text{contours} = \text{FindContours}(M_t^{\text{final}}, \text{RETR\_EXTERNAL}) \tag{22}$$

For each contour with area $A \geq 200$ pixels, the centroid is computed via moments:

$$M_{10} = \sum_{(x,y)\in\text{contour}} x \tag{23}$$

$$M_{01} = \sum_{(x,y)\in\text{contour}} y \tag{24}$$

$$M_{00} = \sum_{(x,y)\in\text{contour}} 1 \tag{25}$$

$$c_x = \frac{M_{10}}{M_{00}}, \quad c_y = \frac{M_{01}}{M_{00}} \tag{26}$$

4.2.3 5. Track Matching

Identical to Implementation 1 (Algorithm 2).

4.3 Key Differences from Implementation 1

5 Comparison of Methods

5.1 DBSCAN vs. Contour Detection

5.1.1 DBSCAN Clustering Advantages

- Detects clusters of arbitrary shape

Aspect	From Scratch	With Libraries
Object Detection	DBSCAN Clustering	Contour Detection
Distance Metric	L_2 Norm (Euclidean)	N/A (contour moments)
Min Area	50 pixels	200 pixels
Blur Kernel	$(15, 15)$	$(15, 15)$
Motion Threshold	20	25
Morphological Ops	Opening + Dilation	Dilation only

- More robust to noise and gaps in motion regions
- Explicitly uses distance metric (Euclidean L_2 norm)
- Handles overlapping/touching objects better
- No predefined number of clusters needed

5.1.2 DBSCAN Clustering Disadvantages

- Computationally more expensive (especially for large images)
- Requires parameter tuning: ϵ and min_samples
- Results sensitive to these parameters
- Custom implementation slower than optimized libraries (e.g., scikit-learn)

5.1.3 Contour Detection Advantages

- Much faster (optimized C++ in OpenCV)
- Simple to implement and understand
- No parameter tuning needed (beyond area threshold)
- Produces connected component centroids directly
- Efficient for well-separated objects

5.1.4 Contour Detection Disadvantages

- Only detects connected components
- Fails for objects with internal holes or gaps
- Less robust to noise
- Assumes binary foreground/background
- Cannot distinguish overlapping objects

5.2 Distance Metrics and Norms

5.2.1 Euclidean Norm (L_2)

Used in DBSCAN clustering:

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{27}$$

- Most natural for Cartesian coordinates
- Rotationally invariant
- Computationally moderate cost
- Physically meaningful (distance in pixels)

5.2.2 Manhattan Norm (L_1)

Alternative that could be used:

$$d(p_i, p_j) = |x_i - x_j| + |y_i - y_j| \tag{28}$$

- Cheaper to compute
- More robust to outliers than L_2
- Less natural for spatial clustering

5.2.3 Derivative-Aware Norms

Proposed enhancement: Use velocity information in clustering

$$d_{\text{vel}}(p_i, p_j, t) = \sqrt{\alpha \cdot \|p_i - p_j\|^2 + \beta \cdot \|v_i - v_j\|^2} \tag{29}$$

where v_i is the velocity at position p_i, and α, β are weights.
Effect on Clustering:

- Objects with similar motion are clustered together
- Better separation of moving objects
- Can distinguish stationary objects from moving ones
- Reduces false associations due to proximity alone
- Requires velocity estimation (adds computational cost)

6 Application Domains and Limitations

6.1 When Methods Work Well

6.1.1 Scenario A: Single Moving Object

- **Conditions**: Clear background, good contrast, steady motion
- **Success**: Both methods perform well
- **Reason**: Motion detection cleanly separates object from background
- **Example**: Skier on snow, person walking against wall

6.1.2 Scenario B: Multiple Well-Separated Objects

- **Conditions**: Objects don't touch, distinct motion patterns
- **Success**: Both methods work; contour method is faster
- **Reason**: Each object produces separate motion region
- **Example**: Multiple vehicles on highway

6.2 When Methods Fail

6.2.1 Scenario C: Touching or Overlapping Objects

- **Condition**: Objects in contact, merged motion regions
- **Failure**: Contour method: treats as single object
- **Failure**: DBSCAN: depends on parameters; can be separated if epsilon chosen well
- **Limitation**: No depth information; ambiguous from 2D projection

6.2.2 Scenario D: Camouflaged Objects

- **Condition**: Object color similar to background
- **Failure**: Motion detection produces weak signal
- **Failure**: Both methods miss or fragment the object
- **Limitation**: Frame differencing requires sufficient contrast

6.2.3 Scenario E: Very Fast Motion

- **Condition**: Object moves several times image resolution between frames
- **Failure**: Frame differencing produces disconnected blobs
- **Failure**: Track matching fails due to large jumps
- **Solution**: Increase $D_{\max}$ or use motion prediction

6.2.4 Scenario F: Shadows and Reflections

- **Condition**: Shadows move independently of objects
- **Failure**: False motion detections
- **Effect**: Tracks spawned for stationary shadows
- **Solution**: Use HSV color space, temporal smoothing

6.2.5 Scenario G: Illumination Changes

- **Condition**: Camera exposure changes, day/night transition
- **Failure**: Large-scale motion detected globally
- **Failure**: All pixels flagged as motion
- **Solution**: Adaptive thresholding, histogram equalization

7 Thresholding and Smoothing Techniques

7.1 Motion Thresholding

The motion threshold τ filters noise in frame difference:

$$M = \begin{cases} 255 & \text{if } |I_t - I_{t-1}| > \tau \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

Effects:

- $\tau = 20$ (from scratch): Sensitive, detects subtle motion, more noise
- $\tau = 25$ (with libraries): Balanced sensitivity and specificity
- $\tau = 50$: Conservative, only strong motion, may miss slow objects

7.2 Morphological Smoothing

7.2.1 Gaussian Blur

Applied before frame differencing:

$$I^{\text{blurred}}(x, y) = \sum_{i,j} G(i, j) \cdot I(x + i, y + j) \tag{31}$$

Kernel size $(15, 15)$ is chosen to:

- Reduce camera noise
- Smooth object edges
- Reduce false motion from noise

7.2.2 Morphological Opening

Removes small noise components:

$$\text{OPEN}(M) = \text{DILATE}(\text{ERODE}(M, K), K) \tag{32}$$

Erode removes small foreground; dilate restores object size.

7.2.3 Morphological Dilation

Connects nearby object parts:

$$\text{DILATE}(M, K, n) = \underbrace{\text{DILATE} \circ \cdots \circ \text{DILATE}}_{\text{n times}}(M, K) \tag{33}$$

From Scratch uses $n = 2$ iterations; fills gaps up to ~ 6 pixels.
Effect on Clustering:

- Larger connected components
- Fewer spurious small clusters
- Risk of merging nearby objects (if dilation excessive)

8 Code Structure Comparison

8.1 Implementation 1: From Scratch

```
# Main Components:
# 1. from_scratch_derivatives() - Compute 4 derivative orders
# 2. dbscan_numpy() - Custom DBSCAN clustering
# 3. RobustTracker.process_video() - Main pipeline
# 4. _detect_objects_via_dbscan() - Cluster-based detection
# 5. _update_tracks() - Greedy track matching
# 6. _draw_overlay() - Visualization
# 7. analyze_tracks_wrapper() - Derivative plotting

# Key Parameters:
blur_ksize = (15, 15)
threshold_val = 20
min_contour_area = 50
max_track_distance = 150
patience = 15
dbscan_eps = 10
dbscan_minpts = 3
dbscan_sample_limit = 5000
```

8.2 Implementation 2: With Libraries

```
# Main Components:
# 1. from_scratch_derivatives() - Compute 4 derivative orders (identical)
# 2. RobustTracker (contour-based version)
# 3. _detect_objects_via_contours() - OpenCV contour detection
# 4. process_video() - Main pipeline (similar structure)
# 5. _update_tracks() - Greedy track matching (identical)
# 6. _draw_overlay() - Visualization (identical)
# 7. analyze_tracks_wrapper() - Derivative plotting (identical)

# Key Parameters:
blur_ksize = (15, 15)
threshold_val = 25
min_contour_area = 200
max_track_distance = 150
patience = 15
# (No DBSCAN parameters)
```

9 Computational Complexity

9.1 From Scratch (DBSCAN)

Per frame:

$$T_{\text{preprocess}} = O(WH) \quad \text{(blur + difference)} \tag{34}$$
$$T_{\text{sample}} = O(NP) \quad \text{(if } N > 5000\text{)} \tag{35}$$
$$T_{\text{DBSCAN}} = O(N \log N) \text{ to } O(N^2) \tag{36}$$
$$T_{\text{match}} = O(T \cdot C) \quad (T \text{ tracks}, C \text{ centroids}) \tag{37}$$

where $W \times H$ is frame resolution, N is number of motion pixels.

9.2 With Libraries

Per frame:

$$T_{\text{preprocess}} = O(WH) \tag{38}$$
$$T_{\text{contours}} = O(WH) \quad \text{(optimized C++)} \tag{39}$$
$$T_{\text{moments}} = O(\text{contour perimeter}) \tag{40}$$
$$T_{\text{match}} = O(T \cdot C) \tag{41}$$

Typically: With Libraries $\approx$ 2–5$\times$ faster than From Scratch.

10 Physical Unit Conversion Analysis

10.1 When Conversion is Possible

Conversion to real units requires calibration information. Valid approaches:

10.1.1 Method 1: Reference Object

If a known object (e.g., person, ball, vehicle) is visible:

$$\gamma = \frac{\text{known\_size [m]}}{\text{pixel\_size [pix]}} \tag{42}$$

Assumptions:

- Object is perpendicular to camera
- No perspective distortion
- Focal length doesn't change

10.1.2 Method 2: Camera Calibration

Using intrinsic camera matrix $\mathbf{K}$:

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{43}$$

where f_x, f_y are focal lengths and c_x, c_y is principal point.
For a depth plane at distance Z:

$$\gamma(Z) = \frac{Z}{f_x} \tag{44}$$

Requirement: Multiple views or depth sensor (3D reconstruction).

10.1.3 Method 3: Stereo Vision

With stereo baseline b and disparity d:

$$Z = \frac{f \cdot b}{d}, \quad \gamma = \frac{Z}{f} \tag{45}$$

Requirement: Synchronized stereo camera pair.

10.2 Why Conversion Often Fails

1. **Unknown camera parameters**: Most consumer videos lack calibration data
2. **Varying distance**: Object at unknown, changing depth (z-axis)
3. **Perspective distortion**: Different parts at different depths
4. **Non-perpendicular motion**: Object motion has component along z-axis

5. **No reference scale**: Nothing in scene with known size

Conclusion: Physical unit conversion is **fundamentally ambiguous** for single-camera video without calibration. The factor γ cannot be uniquely determined.

11 Conclusions and Recommendations

11.1 Method Selection

11.1.1 Use From Scratch (DBSCAN) When

- Objects have complex, irregular shapes
- Objects may touch or overlap slightly
- Robustness to noise is critical
- Computational resources are available
- Need fine control over clustering behavior

11.1.2 Use With Libraries (Contour) When

- Speed is priority
- Objects are well-separated
- Simple blob tracking is sufficient
- Resources are limited (embedded systems)
- Integration with other OpenCV tools

11.2 Recommended Workflow for Computational Project

1. **Problem A (Single Object)**:
    - Use well-controlled video (steady background)
    - Apply With Libraries approach (faster iteration)
    - Verify derivative calculations manually
2. **Problem B (Multiple Objects)**:
    - Use From Scratch with tuned ϵ parameter
    - Test derivative-aware norm for better clustering
    - Compare results with and without dilation
3. **Success Case**:
    - Clean video, high contrast, minimal occlusion
    - Document parameter choices

- Present derivative plots showing smooth curves

4. **Failure Case**:
   - Challenging video (overlapping objects, shadows, fast motion)
   - Document why method fails
   - Propose solutions (e.g., depth-based separation)

11.3 Limitations of Current Approaches

1. **2D projection**: Cannot separate overlapping objects
2. **No scene model**: Cannot use dynamics/physics
3. **Motion-only detection**: Fails for stationary objects
4. **No appearance model**: Cannot handle similar-colored objects
5. **Frame rate dependent**: Accuracy degrades with slow fps
6. **Scale ambiguity**: Cannot convert to real units without calibration

Appendix: Implementation Specifications

Input/Output

- **Input**: MP4 video file (`output.mp4` for From Scratch, `conveir_belt.mp4` for With Libraries)
- **Output (Video)**: `processed_video.mp4` with tracked objects and IDs
- **Output (Plots)**: PNG files in `tracking_results/` folder
- **Tracks Data**: Stored in `tracker.tracks` dictionary

Data Structures

```
# tracks: {track_id: [list of (x, y) centroids]}
# track_patience: {track_id: current patience counter}
# positions: list of (x, y) coordinates

# Example:
tracks = {
    0: [(100, 200), (102, 205), (105, 210), ...],
    1: [(300, 150), (305, 155), (310, 160), ...],
}
```