Nika Koghuashvili
KIU
Numerical Programming
Computational Project 1

Object Tracking and Higher-Order Motion Analysis in Video

1. Problem Statement and Goals

The goal of this project is to implement simple object recognition and motion analysis in video without using any pretrained models. Given a video with a static, stationary background and one or more moving objects, we want to:

- Detect and track each moving object over time.
- Compute its kinematic quantities in pixel units:
  * Velocity (first derivative of position, px/s)
  * Acceleration (second derivative of position, px/s^2)
  * Jerk (third derivative of position, px/s^3)
  * Jounce (fourth derivative of position, px/s^4)
- Optionally convert these to physical units (m/s, m/s^2, ...) when calibration information is available.
- Apply clustering in motion-feature space, using norms that incorporate derivatives, to group objects with similar motion patterns.
- Visualize results in an overlay video: for each tracked object, draw a rectangle around it and print its velocity, acceleration, jerk, and jounce.

We implement two versions:

- A from-scratch version where almost all vision logic (background modeling, foreground segmentation, morphology, connected components, tracking, derivatives) is written using only NumPy, and OpenCV is used strictly for video I/O and drawing.
- A library-based version that uses OpenCV background subtraction and scikit-learn DBSCAN clustering to show how built-in tools simplify parts of the pipeline.

2. Algorithm Overview

Both implementations follow the same pipeline:

1) Preprocessing
   - Read frames, convert to grayscale, optionally downscale to speed up processing.

2) Background modeling and foreground segmentation
   - Estimate the static background.
   - Subtract background and threshold to obtain a binary foreground mask.
   - Apply morphological opening and closing to clean noise.

3) Connected components and object detection
   - Detect blobs (connected components) in the binary mask.
   - For each blob, compute centroid and bounding box.

4) Multi-object tracking
   - For each frame, associate detections with existing tracks using a greedy nearest-neighbor rule in pixel space.
   - Unmatched detections start new tracks.

5) Kinematics
- For each track, build time series of (x, y) centroid positions.
- Compute first, second, third, and fourth derivatives with respect to time: velocity, acceleration, jerk, and jounce.

6) Clustering (library version)
- For each track, compute motion summary features: [mean_speed, max_speed, mean_abs_accel, mean_abs_jerk, mean_abs_jounce].
- Cluster tracks using DBSCAN in this feature space, which implicitly defines a norm that includes derivatives.

7) Visualization
- Re-read the original video.
- For each frame, draw a rectangle around each tracked object and annotate kinematic values.
- Write the result into an output mp4 file.

3. Preprocessing and Background Modeling

Each frame is read from the input video using OpenCV VideoCapture. We convert BGR to grayscale using a standard luminance formula:

$$\text{gray} = 0.299 * R + 0.587 * G + 0.114 * B$$

In the from-scratch version this conversion is implemented manually with NumPy, followed by optional downscaling by a factor K (for example 2 or 4) using simple subsampling. This reduces spatial resolution but speeds up subsequent operations roughly by a factor of K^2.

For the from-scratch background model we assume a static camera and static background. We build a reference background B(x, y) by averaging the first N frames F_t(x, y):

$$B(x, y) = (1 / N) * \sum_{t = 0 \text{ to } N-1} F_t(x, y)$$

Given a new frame F_t, we compute an absolute difference

$$D_t(x, y) = |F_t(x, y) - B(x, y)|$$

and threshold it:

$$M_t(x, y) = \begin{cases} 1 & \text{if } D_t(x, y) > \tau \\ 0 & \text{otherwise} \end{cases}$$

This yields a binary mask M_t that marks foreground (moving) pixels. We then apply custom 3x3 erosion and dilation, and combine them as opening plus closing to remove isolated noise and fill small gaps.

In the library-based version we use OpenCV createBackgroundSubtractorMOG2, plus thresholding and built-in morphology. This is more robust to slow illumination changes but follows the same conceptual idea.

4. Connected Components and Bounding Boxes

On the cleaned binary mask we detect 8-connected components. In the from-scratch version we implement a BFS flood-fill over all foreground pixels. For each component we accumulate:

- area = number of pixels
- sum_x, sum_y = sums of x and y coordinates
- min_x, max_x, min_y, max_y = bounding box coordinates

After BFS, components with area >= MIN_AREA are accepted. For each accepted component we compute:

- centroid: (cx, cy) = (sum_x / area, sum_y / area)
- bounding box: (min_x, min_y, max_x, max_y)

In the library version we use cv2.connectedComponentsWithStats, which provides the same information directly.

The centroids are used as measurements for tracking, and the bounding boxes are later scaled back up and drawn as rectangles on the full-resolution video.

5. Greedy Nearest-Neighbor Tracking

We represent each tracked object as a Track structure with fields:

- id integer track id
- frames list of frame indices
- xs, ys lists of centroid coordinates in downscaled pixel space

For each frame we have a set of detection centroids d_1, d_2, ... We associate them to existing tracks using a greedy nearest-neighbor strategy:

1) For each existing track, take its last known position (x_last, y_last).
2) Compute Euclidean distances to all detections.
3) Attach the closest detection within MAX_TRACK_DIST to that track, and mark it as assigned.
4) After all active tracks are updated, any remaining unassigned detections start new tracks.

This simple strategy works well when objects move smoothly between frames and do not intersect or heavily occlude one another.

6. Kinematics: Velocity, Acceleration, Jerk, Jounce

For each track we obtain:

- times t_i = frame_i / fps
- positions x_i, y_i in pixel coordinates

We approximate derivatives using central finite differences for irregular time steps. For a sequence x(t) we compute:

- velocity v_x = dx/dt
- acceleration a_x = dv_x/dt
- jerk j_x = da_x/dt
- jounce s_x = dj_x/dt

and similarly for y. We then compute magnitudes:

```
speed       = sqrt(v_x^2 + v_y^2)
accel_mag   = sqrt(a_x^2 + a_y^2)
jerk_mag    = sqrt(j_x^2 + j_y^2)
jounce_mag  = sqrt(s_x^2 + s_y^2)
```

We summarize each track by:

- mean_speed
- max_speed
- mean_abs_accel
- mean_abs_jerk
- mean_abs_jounce

Why jerk and jounce matter:

- Velocity describes how fast an object moves.
- Acceleration describes how quickly its speed or direction changes.
- Jerk is the rate of change of acceleration. High jerk indicates rapid starts, stops, or sharp turns. It is important in comfort and mechanical stress analysis (for example in vehicles or robots).
- Jounce is the rate of change of jerk. It is extremely sensitive to small fluctuations and therefore reveals very rough or noisy motion. In our case, jounce is also a useful indicator of tracking noise.

7. Clustering in Motion-Feature Space (DBSCAN)

To explore clustering with norms that incorporate derivatives we construct a feature vector for each track:

```
f = [mean_speed, max_speed,
     mean_abs_accel, mean_abs_jerk, mean_abs_jounce]
```

This embeds both position derivatives and their magnitudes in a five-dimensional feature space. Two tracks that have similar speeds but very different jerk or jounce will be far apart in this space.

We standardize the feature matrix and run DBSCAN. DBSCAN groups tracks with similar motion patterns into clusters and labels outliers with -1. The cluster id is displayed along with the track id in the overlay video, for example:

```
ID 3 (C1)  meaning  track 3 in cluster 1.
```

By changing the relative weighting of derivatives in the feature vector we can change the effective norm and therefore the clustering behavior.

8. Conversion to Physical Units

Our kinematic quantities are computed in pixel units:

- v_px in px/s
- a_px in px/s^2
- jerk_px in px/s^3
- jounce_px in px/s^4

If we know a scale factor S in meters per pixel (m/px), for example from the known diameter of the ball or from calibration markers, then we can convert:

- v_real = S * v_px (m/s)
- a_real = S * a_px (m/s^2)
- jerk_real = S * jerk_px (m/s^3)
- jounce_real= S * jounce_px (m/s^4)

This conversion is valid if the motion is approximately in the image plane and the perspective distortion is small. For more complex 3D motion, full camera calibration and 3D reconstruction would be required.

9. Experiments

We tested the method on three types of videos.

9.1 Video A: Single Object, Static Background (Success)

- One bright ball rolling across a clean, static floor.
- Camera fixed on a tripod, no panning or zoom.
- Background is simple and high-contrast.
- No other moving objects.

In this scenario:

- Background subtraction produces a clean mask.
- Connected components detect exactly one blob per frame.
- The greedy tracker produces a single, stable track.
- Velocity, acceleration, jerk, and jounce are smooth and physically reasonable.

9.2 Video B: Multiple Objects (5 or more)

- Several objects (balls or cars) moving on a static track.
- Camera fixed; background is static.
- Objects differ in speed and possibly size.

Here:

- The method detects multiple blobs per frame.
- The tracker successfully maintains separate tracks for most objects, as long as they do not heavily occlude each other.
- The library-based version clusters tracks according to motion features: some clusters correspond to slow and smooth trajectories, others to faster or more jerky ones.

9.3 Video C: Moving Background or Camera Shake (Failure)

We also tested a failure case where:

- The camera slightly moves (hand-held), or
- The background itself is dynamic (people walking, trees moving, strong shadows).

In such cases:

- Background subtraction marks large parts of the image as foreground.
- Connected components include many false blobs.
- Tracks fragment, merge incorrectly, or jump between objects.
- Higher-order derivatives (especially jerk and jounce) become dominated by noise and do not reflect real physical motion.

10. Why It Works and When It Fails

The algorithm assumes:

1) Static camera and static background.
2) Moderate inter-frame motion so that centroids do not jump too far.
3) Limited and short occlusions.
4) Reasonable contrast between objects and background.

In Videos A and B these assumptions largely hold, so the method works well.
In Video C one or more assumptions fail, which breaks background subtraction and data association, and therefore corrupts the kinematic estimates.

Understanding these limitations is crucial: it shows that classical background subtraction plus centroid tracking is well suited to controlled scenes but not to crowded, dynamic, or hand-held camera footage.

11. Conclusions

We have implemented a complete pipeline for multi-object motion analysis using simple computer vision tools:

- From-scratch background subtraction, morphology, connected components, and greedy tracking with NumPy.
- A second version that uses OpenCV background subtraction and scikit-learn DBSCAN for clustering.

We computed position, velocity, acceleration, jerk, and jounce in pixel units, visualized them as overlays on the original video, and showed how derivative-based features can be used in a clustering algorithm.

The experiments demonstrate:

- Strong performance on videos with static backgrounds and moderate motion.
- Clear failure modes when the background or camera moves or when occlusions are severe.

Possible future improvements include:

- Temporal smoothing or Kalman filters to stabilize trajectories and reduce noise in higher-status derivatives.
- More robust data association (for example the Hungarian algorithm) for crowded scenes.
- Full metric calibration to convert from pixels to physical units in meters.