# Final Project Report

# Video Processing Pipeline

Course: Video Processing 24/25

Students Names: Amir Levy & Binat Makhlin

Students IDs : 208484097 318931573

Submission Date: 14.7.25

Git Repository

# Table of Contents

## Problem Statement

The purpose of this project is to implement a complete video processing pipeline designed to process a shaky video containing a single person walking. The objective is to produce a fully processed output video where:

- The camera motion is stabilized.
- The person is extracted from the original background using advanced background subtraction.
- The extracted person is composited onto a new background.
- The person is tracked throughout the video and bounded by a box in each frame.
- All intermediate and final results are saved, including binary masks, alpha mattes, and tracking metadata.

## Algorithm Overview

The processing pipeline includes the following four stages:

1. **Stabilization**

   The stabilization module first converts each frame to grayscale, then uses OpenCV's goodFeaturesToTrack to detect distinctive corner points, which serve as key landmarks. These landmarks are tracked into the next frame with the Lucas–Kanade optical flow method. For each pair of frames the code estimates a 3×3 homography matrix via RANSAC, falling back to the identity matrix if there are too few matches, thus capturing the raw camera motion. All per-frame transforms are cumulatively summed into a trajectory, which is smoothed with a moving-average filter to remove high-frequency jitter. Finally the inverse of each smoothed transform is applied to its original frame using perspective warping and a slight border scale-up, aligning every frame to a common reference and yielding a jitter-free output video.

2. **Background Subtraction**

   The background-subtraction pipeline in background_sub.py separates the person from the background in two main stages: first it applies KNN background subtraction on the HSV frames, using only the saturation and value channels, to produce an initial coarse mask; then it refines that mask with kernel-density color models by sampling foreground and background pixels from the body, shoes and face regions - these samples train separate Gaussian KDE′s, which are applied in a sliding window, merged with morphological opening

and contour filtering to produce the final mask.
A blue-channel threshold filters out poster-like backgrounds before sampling, and height-based ratios define the shoe and shoulder regions for color collection.

- OpenCV's BackgroundSubtractorKNN models (not learned in class):
  each pixel as a sliding buffer of the last N colour samples. For a new pixel value x the algorithm computes the Euclidean distance d(x, $x_i$) to every stored sample $x_i$. If at least K samples lie within a distance threshold D the pixel is labelled background, otherwise foreground.
  After classification a random buffer entry is replaced by x with probability α, therefore the model adapts gradually to slow background changes.
  Typical defaults are N = 120, K = 2 – 4, D derived from pixel variance, α=0.01.
  Compared with the Gaussian-Mixture model taught in Lecture 7, KNN avoids estimating per-component covariances, so memory is lower, initialisation is trivial and adaptation to illumination drift is faster; the drawback is sensitivity to noise and dynamic textures because no statistical weighting is used.
  In this project K = 3, N = 120 and OpenCV's automatic D were kept; morphological closing and median filtering are applied afterwards to suppress isolated false positives.

At the end the script writes two output videos: 'extracted_208484097_318931573.avi', showing the person on a black background, and 'binary_208484097_318931573.avi', containing the binary mask.

## 3. Matting and Compositing

The matting module in matting.py composites the extracted person onto a static background in three steps: first it builds a trimap from the binary mask by eroding and dilating to define definite foreground, definite background, and unknown regions;
next it computes an alpha matte - either with a fast distance-transform method or the closed-form solver - followed by bilateral filtering to smooth transitions;
finally it blends the person pixels and the new background using the alpha matte.
Morphological operations on the trimap and Gaussian-bilateral filtering on the alpha channel ensure clean boundaries and minimal artifacts.

- Closed Form Matting – Algorithm Description (not covered in class)
  Closed Form Matting assumes that within a small three by three window the RGB values of the original image can be approximated by a local line, therefore the opacity map α is expected to vary linearly with the colours inside that window. where **L** is the sparse matting Laplacian. For every window ω with mean colour μ and covariance Σ the Laplacian entries are

$$L_{ij} = \delta_{ij} - \frac{1}{|\omega|} * [1 + (I_i - \mu)^{\mathrm{T}} + \Sigma^{-1} * (I_i - \mu)]$$

We build L once per frame, assemble the diagonal constraint matrix C from the trimap, and solve the linear $(L + \lambda C)\alpha = \lambda C \alpha_{known}$ with scipy.sparse.linalg.spsolve, using $\lambda = 100$.
The result is a high-quality alpha matte that preserves fine hair and soft edges, but the cost is high memory usage and a runtime of several seconds per frame because the matrix dimension equals the number of pixels.

**4. Tracking**

The tracking module in tracking.py begins by selecting a full-body region of interest on the first frame using fixed percentage offsets. It converts that ROI to HSV, computes a two-channel color histogram with a light-reduction mask, and normalizes it.
On each subsequent frame it backprojects the histogram to create a probability image, then applies OpenCV's meanShift with predefined termination criteria to update the bounding window.
The code draws the resulting rectangle on each frame and records the coordinates as [y, x, height, width] in a dictionary keyed by frame index. Finally it writes out the annotated video and saves the tracking data to a JSON file.

## Implementation Details

- Each stage of the pipeline is implemented in a separate Python module.
- The pipeline is executed sequentially from a single entry point (main.py).
- All file paths are relative, and the directory structure follows the course specification.
- The implementation uses Python 3.11, with OpenCV, NumPy, and SciPy.
- Runtime statistics are logged to timing.json as required.
- Pipeline runs fully automatic, no GUI windows, all paths are relative, total runtime ≤ 20 min on Tochna lab PCs (Intel i9-10900, 16 GB). No deep-learning libraries used.

## Design Decisions and Enhancements

Several design choices were made to improve accuracy and generalization:

- KDE models are computed dynamically from per-frame color samples rather than fixed heuristics.
- Morphological operations (e.g., closing and opening) are applied with adaptive kernel sizes for different regions.

- Shoes and face regions are handled separately to account for their distinct color distributions.
- A caching mechanism is used to speed up KDE evaluations across frames.
- Face region extraction is restricted to the central portion of the upper mask to avoid background interference.
- A blue-channel threshold is applied before sampling to suppress dominant background colors (e.g. poster walls) so only the person's colors are modeled.
- A sliding-window around the mask's centroid is used when applying the KDEs, ensuring the color models focus on the local region surrounding the person rather than the entire frame.
- Closed-form matting is the default for quality, guided filter is offered when runtime is critical, switchable via method='optimized'.

## Results

The processed video successfully demonstrates the following outcomes:

- The video was stabilized, reducing camera shake.
- The person was extracted effectively, including shoes and head regions.
- KDE-based filtering reduced background leakage.
- The person was composited over a new background.
- Bounding boxes were correctly applied to each frame and exported.
- Intermediate videos include stabilized, binary mask, alpha matte, extracted foreground, and final output.

## Files and Outputs

The submission is a single archive named 'FinalProject_208484097_318931573.zip'.

Inside the archive there are five mandatory folders, plus one optional working folder:

- Document/ holds the PDF report you are reading.
- Code/ contains every Python script, headed by main.py, together with requirements.txt. All imports are relative and no absolute paths appear in the code.
- Input/ stores the two raw inputs exactly as provided: INPUT.avi and background.jpg.
- Outputs/ is created automatically by the pipeline and contains six colour videos and two JSON files, all tagged with the two ID numbers and written by main.py in this order:
    - stabilized_208484097_318931573.avi
    - extracted_208484097_318931573.avi

  - binary_208484097_318931573.avi (grayscale)
  - alpha_208484097_318931573.avi
  - matted_208484097_318931573.avi
  - OUTPUT_208484097_318931573.avi
  - timing.json
  - tracking.json

  The timing.json file records, for each generated video, the elapsed seconds since the start of execution. The tracking.json file maps every frame index to [row, col, height, width] of the bounding box.

- ScreenRec/ includes a screen-capture video run_screen.mp4 that shows the entire pipeline executing at two-times speed, proving the run is fully automatic and finishes in less than twenty minutes on a Tochna lab workstation.

Every path inside the code is constructed with os.path.join, therefore the project can be unzipped to any location and run with the single command :python Code/main.py without further setup.

## Future Work

- Improve segmentation of arms and fingers through motion cues.
- Implement temporal mask smoothing across frames.
- Apply spatial smoothing (e.g. morphological or bilateral filtering) specifically to the shoe and face regions to reduce local artifacts.
- Experiment with Poisson blending for more natural compositing.
- Integrate optical flow into face tracking to stabilize mask alignment.

## References


- OpenCV Documentation
- SciPy KDE API: scipy.stats.gaussian_kde
- Video Processing course materials and assignment slides