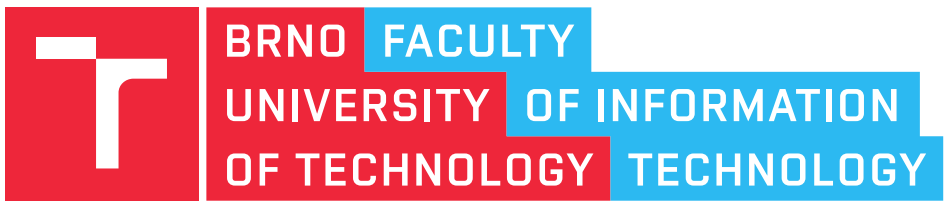
Image stabilization

Guilherme Franco & Miguel Rodrigues

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
{xfranc01, xboave00}@fit.vutbr.cz



January 2024

Task definition

The aim of the project is to create a video stabilization app, capable of compensating for unwanted camera movement.

Task definition

The aim of the project is to create a video stabilization app, capable of compensating for unwanted camera movement.

There are two sets of methods used to solve video stabilization:

- **Classical methods** follow a "three-step" approach:
  1. Motion estimation.
  2. Motion compensation.
  3. Image Warp.
- **Deep learning methods** based on convolution neural networks (CNNs):
  - Computationally more demanding.
  - Learn image transformations.

Video stabilization methods

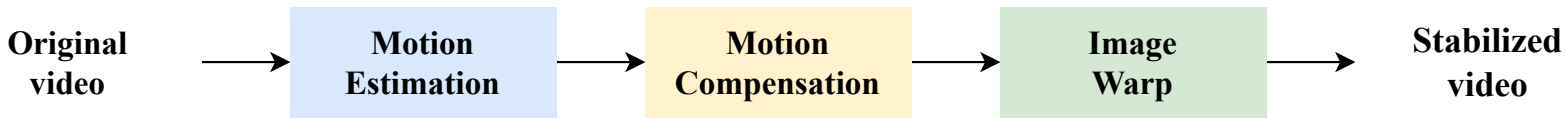


Figure: Classical video stabilization methods framework. Taken from (1).

Video stabilization methods

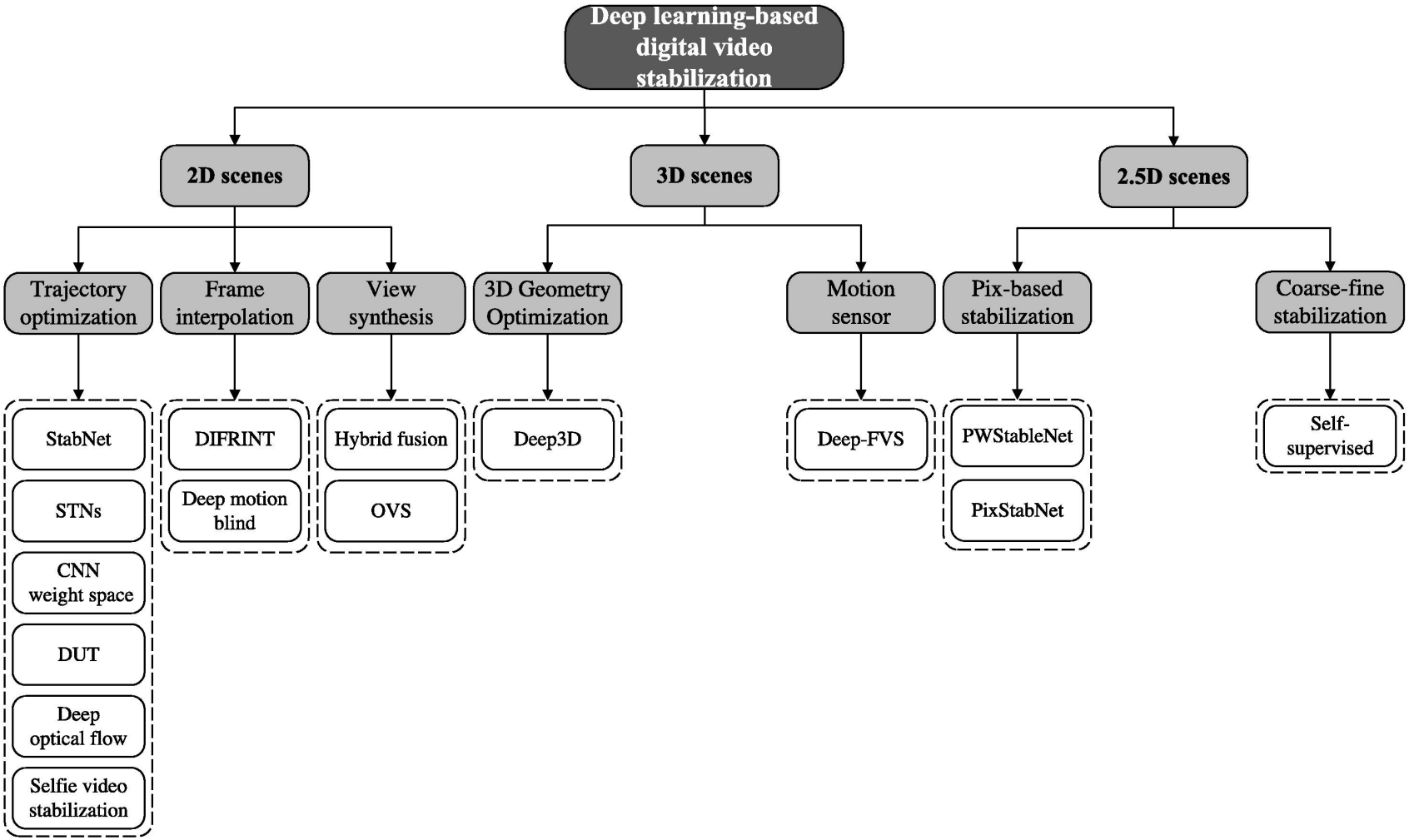


Figure: Topology of deep learning based methods. Taken from (1).

We opted to follow a classical approach.

- *"Focus on the speed of stabilization."*

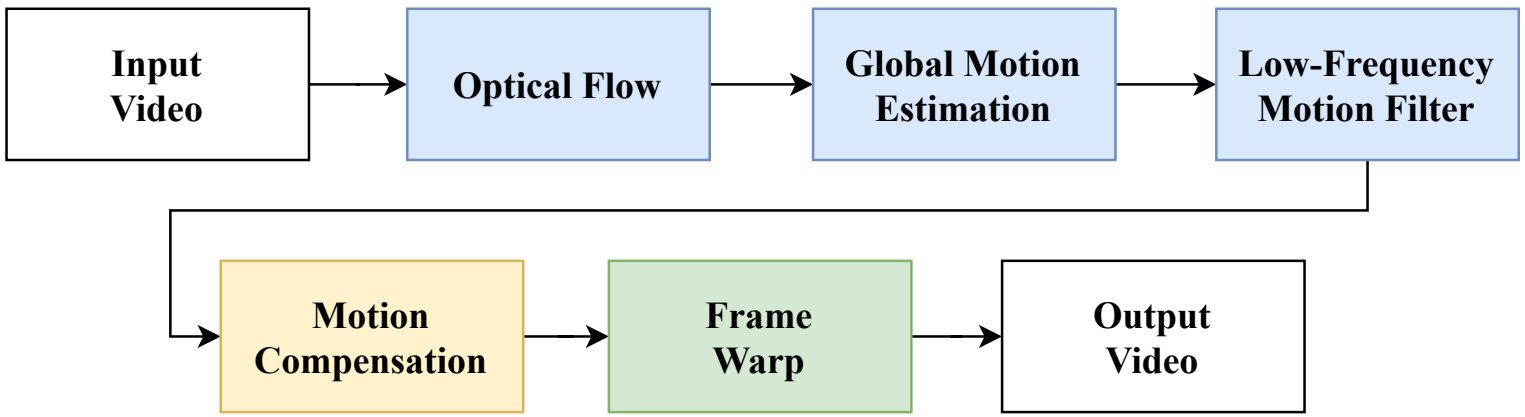


Figure: Our implementation's pipeline.

Optical flow

- Our first attempt was using PyTorch's RAFT (2) algorithm:
  - State of the art algorithm based on deep architecture.
  - Dense method with a great accuracy.
  - **Very slow**, processing 15 frames took around 2 minutes!

Optical flow

- Our first attempt was using PyTorch's RAFT (2) algorithm:
  - State of the art algorithm based on deep architecture.
  - Dense method with a great accuracy.
  - **Very slow**, processing 15 frames took around 2 minutes!
- The final solution uses Lucas-Kanade from OpenCV's `cv2.calcOpticalFlowPyrLK()` (3):
  - Sparse methods that depends on a set of features.
  - Features obtained with `cv2.goodFeaturesToTrack()` using the Shi-Tomasi (4) corner detection method.

After the optical flow stage, **transformations** between frames are computed.

- We assumed an *euclidean* motion model, admitting only **translation** and **rotation**.
- Transformation is an estimation of the frame movement using features that matched in both frames:
  - `cv2.estimateAffinePartial2D()`, from OpenCV computes these transformations.
  - It is robust to outliers as it uses RANSAC.
  - It has some troubles with frames with low brightness.

The result of the global motion estimation is the triple (x, y, θ). Obtained from the estimation matrix T' using the relations:

$$x = T'_{01} \tag{1}$$

$$y = T'_{02} \tag{2}$$

$$\theta = \arctan \frac{T'_{10}}{T'_{00}} \tag{3}$$

The goal of motion smoothing is to **remove the unwanted camera movement**.

The goal of motion smoothing is to **remove the unwanted camera movement**.

- An analogy is to think about movement as a signal:
  1. Unwanted camera movement is high-frequency movement.
  2. Apply a low-pass filter keeping slow and steady movement.

Motion Smoothing

The goal of motion smoothing is to **remove the unwanted camera movement**.

- An analogy is to think about movement as a signal:
  1. Unwanted camera movement is high-frequency movement.
  2. Apply a low-pass filter keeping slow and steady movement.
- A transformation represents the velocity of the camera.
  - The trajectory of the camera is its integral (cumulative sum).
  - Then, trajectories are filtered along each component.
  - In our solution, we opted for a simple **moving average** filter.

After filtering the trajectories, motion compensation is applied to each frame.

The smoothed transformations are described by:

$$F_{smooth} = F_{original} + (J_{smooth} - J_{original}) \tag{4}$$

- F is a transformation between consecutive frames.
- J is the trajectory between consecutive frames.

The next step is to **warp the frame to the right place** using the smoothed transformation matrix.

1. For each frame construct a new matrix T from (x, y, θ):

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & x \\ \sin\theta & \cos\theta & y \end{bmatrix} \tag{5}$$

2. Call `cv2.warpAffine()` function from OpenCV.
3. Apply some scaling, i.e. cropping, to remove the black borders originated after the warping.

To measure the The effectiveness of our video stabilization solution, we have gathered several results:

- Trajectories evolution over the course of the video.
- Time taken to process each video.
- Qualitative assessment.

`https://youtu.be/gEne8OERehE`

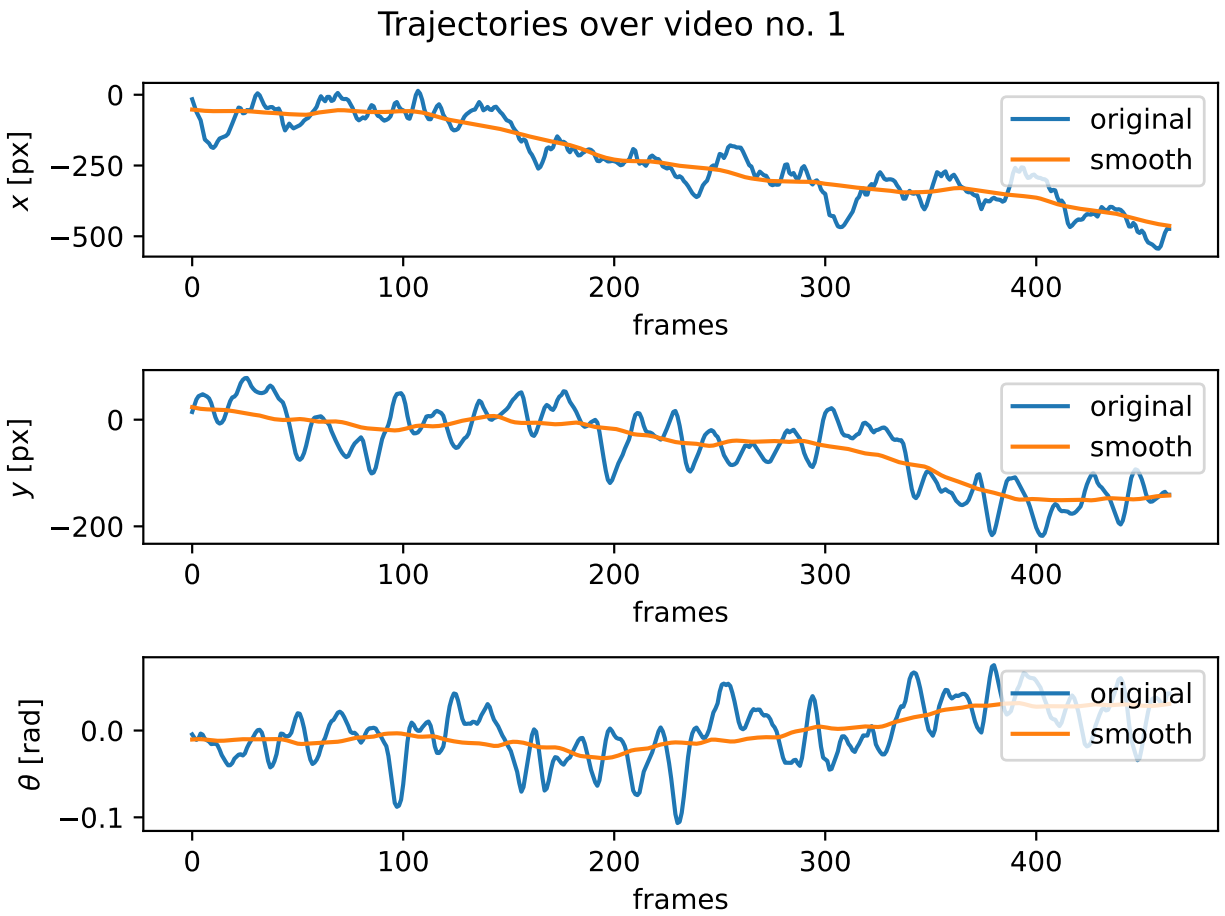


Figure: Trajectories over the course of video no. 1 from the DeepStab dataset.

- Our implementation takes linear time w.r.t. to the length of the video, i.e., number of frames.
- It takes, on average, 17 milliseconds to process a frame when running on an Apple M2 chip.

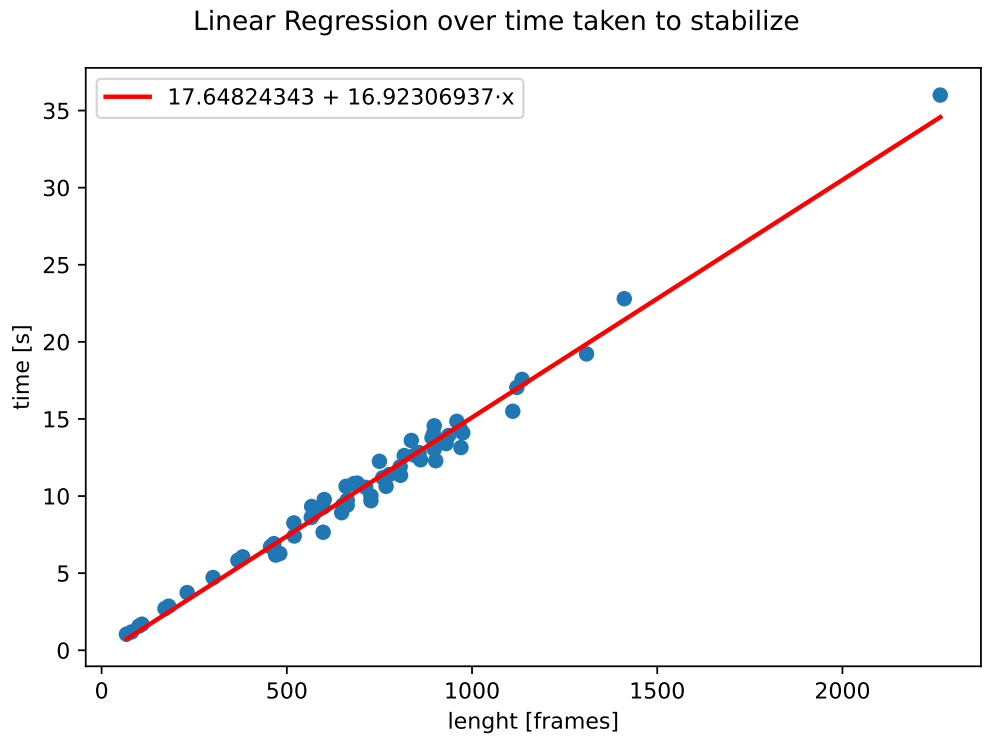


Figure: Line fitting between video length and time to process. Each point represents a unstable video from the DeepStab dataset.

Even though our solution is pretty decent, there are some problems and experiments that were left out.

Some relevant future work would be:

- Use different and more advanced methods of feature detection and/or optical flow.

Even though our solution is pretty decent, there are some problems and experiments that were left out.

Some relevant future work would be:

- Use different and more advanced methods of feature detection and/or optical flow.
- Discard the optical flow computation and obtain transforms based on the ECC criterion (5).
  - `cv2.findTransformECC()` in OpenCV.
  - Uses pixel brightness to find transforms.
  - Commonly employed in image registration.

Even though our solution is pretty decent, there are some problems and experiments that were left out.

Some relevant future work would be:

- Use different and more advanced methods of feature detection and/or optical flow.
- Discard the optical flow computation and obtain transforms based on the ECC criterion (5).
  - `cv2.findTransformECC()` in OpenCV.
  - Uses pixel brightness to find transforms.
  - Commonly employed in image registration.
- Work on deblurring and wobbling effect removal.
  - Most implementations are not efficient enough for our work and only work with gray images.

Thank You For Your Attention !

References I

- Y. Wang, Q. Huang, C. Jiang, J. Liu, M. Shang, and Z. Miao, "Video stabilization: A comprehensive survey," *Neurocomputing*, vol. 516, pp. 205–230, 2023.
- Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16*, pp. 402–419, Springer, 2020.
- J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," 1999.
- J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pp. 593–600, IEEE, 1994.
- G. D. Evangelidis and E. Z. Psarakis, "Parametric image alignment using enhanced correlation coefficient maximization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 10, pp. 1858–1865, 2008.