Video Stabilization Algorithm

By Eden Leyba

The algorithm is an implementation of digital video stabilization, which means no special sensors are required to execute it. It can be divided into 3 stages:

1. Motion estimation – Estimating the change between two consecutive frames in the video.
2. Motion smoothing – Filtering out unwanted movements.
3. Image composition – from stages 1 and 2.

The algorithm is using the Euclidean motion model, by applying Euclidean transformation incorporating changes in location, angle and size.

The algorithm is implemented with the help of OpenCV – an open-source framework for Computer Vision, and NumPY.

Step 1: Set Input and Output Videos

Use the OpenCV framework for defining the setup of input and output videos.

Step 2: Convert the first frame to grayscale

Read the first frame and convert it to grayscale.

Step 3: Find motion between frames

Iterate Over all the frames, convert each one to grayscale and find motion between the current and the previous frame.

In order to do that, we need to first find “good” (=feature) points in each frame to track and use it to the next frame. That way we can tell the motion of the object in the frame.

3.1 Find good feature points

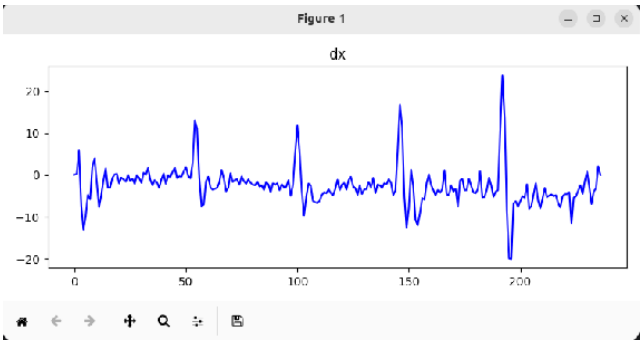
Using the method **goodFeaturesToTrack** of the OpenCV library. The Euclidean model allows us to only know the motion of 2 points in the two frames, but to make better estimations we use a maximum of 200 points (Because not all feature points we found in the first frame are relevant in the next frame).

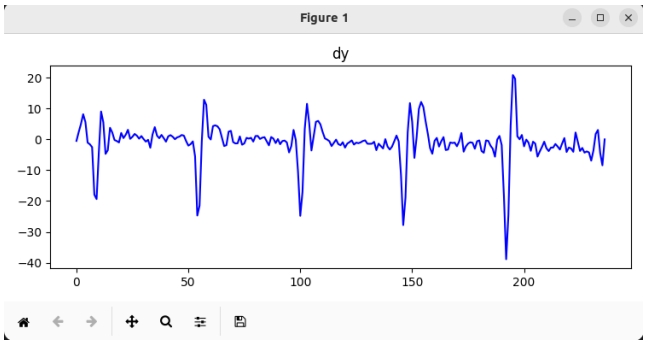
3.2 Track the points

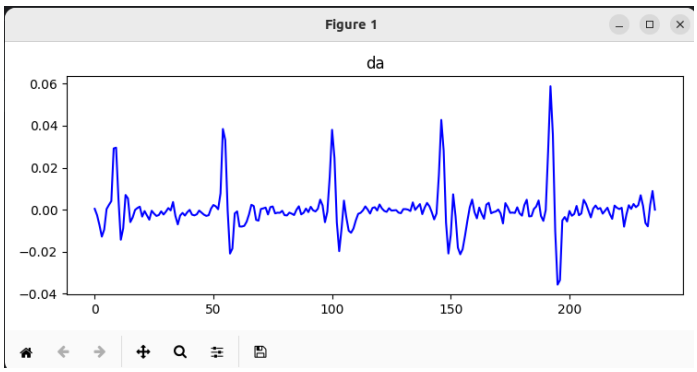
We can track the points from the previous frame using an algorithm called **Lucas-Kanade Optical Flow**. We use a status flag to know if the feature point is relevant in the next frame and filter out only the relevant ones. Now we know the location of the feature points in the current frame and the next frame.

3.3 Estimate Motion

Find the transformation matrix (מטריצה מייצגת) of the transformation that maps the previous frame to the current frame. We do that using the function **estimateAffinePartial2D**, because **estimateRigidTransform** is deprecated. (Partial only uses 4 degrees of freedom, the non-partial one uses 6). In this stage we define the *transforms* array, which is row has 3 entries: dx, dy and da, which are changes to the x coordinate, the y coordinate, and the angle, respectively. Graphs of dx, dy and da are shown below (for example video given at the assignment description).







Step 4: Calculate smooth motion between frames

Finding the trajectory of motion of all of the frames.

4.1 Calculate trajectory

In this step, we will add up the motion between the frames to calculate the trajectory. Our ultimate goal is to smooth out this trajectory.

4.2 Calculate smooth trajectory

Using the moving average filter, we can smooth the curve. If *c* is the array of transformations, then we calculate the filter at the k-th element of the curve by using:

$$f[k] = \frac{c[k-2] + c[k-1] + c[k] + c[k+1] + c[k+2]}{5}$$

To do that we implemented the functions **smooth** which applies for each point the **moving_average** function, which applies the filter mentioned above to a given curve.

4.3 Calculate smooth transforms

We will use the smooth trajectory to obtain smooth transformations that can be applied to frames of the videos to stabilize it.

Step 5: Apply smoothed camera motions to frame

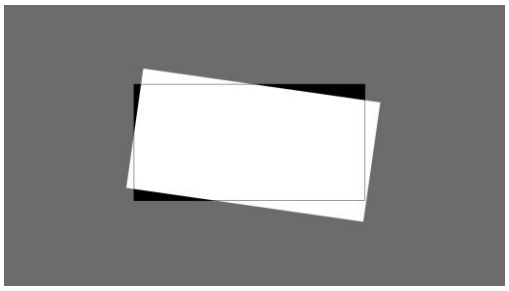
If we have a motion specified as (x, y, θ) , the corresponding transformation matrix is given by:

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \end{pmatrix}$$

.

We implement and use the function **fixBorder,** which fixes the following problem:

Essentially what we are doing is applying changes in location, angle and size. That can lead sometimes to empty, black areas in the output. Here is an illustration for better understanding:



The gray area is the background. The Black is the canvas of the frame, and the white area is the frame itself. Because we rotated and changed the location of the frame, we have “dead” regions in the canvas.

We can fix that by scaling the frame from the center. The value we chose was 20%, after experimenting. This is implemented using **getRotationMatrix2D** of the OpenCV library. All we need to do is call this function with 0 rotation and scale 1.20 (i.e. 20% upscale).