Final Report for L1 Optimal Camera Paths Video Stabilization Project

Omar Hamdache
22001398
Computer Engineering

Furkan Çelik
21901792
Electrical and Electronics Engineering

***Abstract*—In this project, we aim to obtain a stabilized video by eliminating unwanted camera motions. For that purpose, instead of just suppressing high-frequency motion paths, we employed the method mentioned in the article Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths [1]. The methodology mentioned in this paper mainly consists of 3 steps. First, the original motion in the original video is estimated and denoted as C_t. Secondly, the stabilized camera trajectory P_t = C_t * B_t is estimated by finding the proper stabilizing B_t transform. To find B_t, the linear programming solution method was utilized to minimize the objective function consisting of the weighted sum of the first, second, and third derivatives of the resulting camera path subject to some constraints. Finally, the stabilized video is synthesized using the estimated smooth camera path.**

I. Introduction

Our method operates in three main steps: firstly estimating the original shaky camera path, secondly estimating a new smooth camera path, and thirdly reproducing the stabilized video using the estimated smooth camera path.

For that purpose, we first found the good features in the video frames to track and used them in the Lucas-Kanade optical flow algorithm to produce the transformation matrix that explains the linear motion between each frame. This parametric linear motion model F_t, is iteratively multiplied at each frame to obtain the original camera path C_t. Given the original path C_t, the desired smooth path is expressed as P_t = B_t * C_t where B_t is the update transform when applied to the original video path C_t, yields the optimal path P_t. To obtain B_t, the objective function,

$$O(P) = w_1|D(P)|_1 + w_2|D^2(P)|_1 + w_3|D^3(P)|_1 \quad (1)$$

subject to inclusion and proximity constraints is solved by the Linear Programming method.

II. Implementation

In this section, we examine our methodology and the parts we have implemented.

A. Features to Track

Since the algorithm used estimates the motion of the camera by tracking the movements of the feature points between consecutive frames, the first step should be to select good features to track.

For that purpose after reading each frame of the video, we used the `goodFeaturesToTrack` method of `openCV` library.

The function `cv.goodFeaturesToTrack` uses the Shi-Tomasi algorithm which is a one-step advanced version of the Harris corner detection algorithm. The Shi-Tomasi algorithm computes the Hermitian matrix for the given block of pixels and chooses the minimum of the eigenvalues of the Hermitian matrix as scoring. As the eigenvalues increase, the quality of the corner increases [2]. The parameters of the function are set to: `maxCorners=200`, `qualityLevel=0.015`, `minDistance=30`, and `blockSize=20`.

$$H = \begin{bmatrix} I_x^2 & I_xI_y \\ I_yI_x & I_y^2 \end{bmatrix} \quad (2)$$

$$Hx_+ = \lambda_+ x_+$$
$$Hx_- = \lambda_- x_-$$

In the equation (1), H is the Hermitian matrix and (λ) s are the eigenvalues of it [3].

$$R = min(\lambda_1, \lambda_2) \quad (3)$$

The equation (2) gives the Shi-Tomas scoring function.

In this method, there are some parameters that we played to optimize the selection of features. For instance, increasing the quality level enabled us to track better corners, rejecting the points lying in the flat region. Also, we can choose the number of corners to keep track of and the block size of the pixels that the algorithm searches for corners inside.

Another method tried to obtain good features was an object detection algorithm to get more quality feature points in a specific area. For that purpose, the face-tracking algorithm of the `mediapipe` library of the python was utilized [4]. This algorithm provides around 468 3D face landmarks from specific regions of the face and then, these feature points are fed to the optical flow algorithm. In this way, it was observed that the deformations in the specific region of interest can substantially be eliminated, and better stabilization can be obtained.

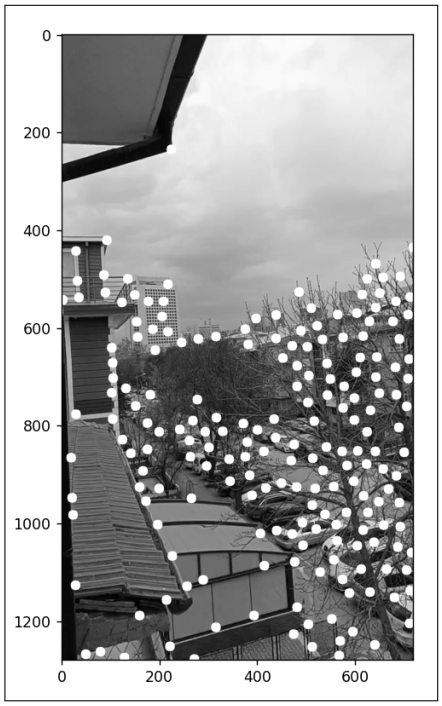

Fig. 1. The selected feature points (corners) to be tracked.

Fig. 2. The provided feature points on the face.

B. Optical Flow

After selecting good features to keep track of, the positions of these points are fed to the Lucas-Kanade optical flow algorithm. In our optical flow implementation, we used `calcOpticalFlowPyrLK` method from the python `cv2` library [5]. In this algorithm, a block of points is chosen around the specified feature points, and the corresponding movement of that block between the consecutive frames is computed. Hence, if a flow is found, the algorithm gives the position of the points that we are tracking in the new frame. Therefore, we can draw lines between these points to represent the flow.

Fig. 3. The calculated optical flow of feature points.

As can be seen from the image, there are some points that give incorrect flows. Especially the points that are misclassified as corners give erroneous flows which is expected in the Lucas-Kanade optical flow algorithm. The reason for that is, that the Lucas-Kanade optical flow algorithm uses the Hermitian matrix to calculate the flow of the block of points in the image assuming brightness constancy. Hence, this algorithm gives better results for the flow of corners. To eliminate the erroneous results we have, we optimized the parameters of the goodFeaturesToTrack method to have more quality features.

C. Affine Transformation

The next step is to model the movement of those feature points so that we can use them to construct a camera path $C(t)$ later. This movement of the feature points can be modeled by using a matrix F_t that describes the motion of the feature points from the frame I_t to I_{t-1}. We use a 6 Degrees of Freedom (DOF) affine transformation to estimate the camera's motion across frames by calculating the affine transformations that best describe the movement between each pair of

consecutive frames. The 6 DOF refers to the combination of translation, rotation, scaling, and shearing transformations that can be applied to a specific frame to reach the next one. F_t can be written as

$$F_t = \begin{pmatrix} a_t & b_t & \Delta x_t \\ c_t & d_t & \Delta y_t \\ 0 & 0 & 1 \end{pmatrix}^T \tag{4}$$

where:

- a_t and d_t represent scaling factors along the x and y axes, respectively,
- b_t and c_t are shearing factors,
- Δx_t and Δy_t are translations along the x and y axes.

In our code, we use `estimateAffine2D` function from the python `cv2` library. The function calculates a transformation matrix (F_t described before) that best describes the motion between the previous and current frames based on the tracked feature points. The function `estimateAffine2D` takes as input the points of the current frame t and the previous frame $t-1$ that were extracted using the optical flow algorithm described in the previous section. The transformation matrix for each frame is stored in the `F_transforms` array. This array builds up the piece-wise representation of the camera's movement across the video.

Below is an example of the output of the variables `prev_pts` (the feature points of the previous frame), `curr_pts` (the feature points of the current frame), and `F_t` (the first two rows of the F_t matrix computed using the `estimateAffine2D` function).

$$prev\_pts = \begin{pmatrix} (824 & 367) \\ (694 & 142) \\ (683 & 249) \\ (1095 & 612) \\ (696 & 102) \\ \cdots & \\ \cdots & \end{pmatrix}$$

$$curr\_pts = \begin{pmatrix} (798.5739 & 366.65085) \\ (669.0923 & 145.35805) \\ (658.46436 & 250.9622) \\ (1065.6102 & 604.47644) \\ (671.3517 & 106.01224) \\ \cdots & \\ \cdots & \end{pmatrix}$$

$$F_t = \begin{pmatrix} 1.01083e+00 & 8.23656e-03 & 1.67490e+01 \\ 2.27715e-03 & 1.01603e+00 & -1.19705e+01 \end{pmatrix}$$

D. Camera Path and Feature Points Motion

As mentioned in the paper, the camera path C_t is computed by the right multiplication of the F_t matrices that we computed previously [1]. More precisely, we have

$$C_{t+1} = C_t F_{t+1} \implies F_1 F_2 ... F_t \tag{5}$$

where we compute C_t from the previously calculated transformation matrices F_t between each two frames. The computed camera path C_t will contain the unstable path that the camera followed during the video.

After following the previously described steps we obtain a camera path C_t, which "describes" how the camera is moving during the footage, by tracking some points on the frames and modeling their movement using transformation matrices F_t.

E. Linear Programming Optimization Problem

In this part, the objective function consisting of the weighted sum of the derivatives of the optimal path is minimized.

The objective function to be minimized:

$$O(P) = w_1|D(P)|_1 + w_2|D^2(P)|_1 + w_3|D^3(P)|_1 \tag{6}$$

where $|D(P)|_1$ represents the camera's path absolute change in position between frames, $|D^2(P)|_1$ represents the camera path's acceleration, and $|D^3(P)|_1$ represents the change in acceleration, or jerk, in the camera motion. Solving this optimization problem will lead us to the smooth camera path P_t. The constant weighting factors w_1, w_2, and w_3 respectively suppress the velocity, acceleration, and jerk. Increasing the weights leads the optimization to decrease the L1-norm of the corresponding derivative. Therefore, with properly chosen weights, this equation leads to a smooth trajectory. Since the undesired effects of shaky video mainly stem from the jerk, the w_3 was chosen the highest to eliminate the effects of jerk more. As suggested by the paper, w_1 was chosen 10, w_2 was chosen 1, and w_3 was chosen 100.

The residuals $|D(P)|_1$, $|D^2(P)|_1$, and $|D^3(P)|_1$ can be expressed in the following ways:

1. $|D(P)|_1$: By the Taylor series expansion, the forward differencing of the frames gives the first derivative of the optimal path in the form

$$|D(P)| = \sum_t |P_{t+1} - P_t| = \sum_t |C_{t+1}B_{t+1} - C_t B_t|$$

Applying the decomposition of C_t:

$$|D(P)| = \sum_t |C_t F_{t+1} B_{t+1} - C_t B_t| \leq \sum_t |C_t||F_{t+1}B_{t+1} - B_t|$$

With C_t known, we therefore seek to minimize the residual:

$$\sum_t |R_t|, \text{ with } R_t := F_{t+1}B_{t+1} - B_t$$

2. $|D^2(P)|_1$: By the Taylor series expansion, the forward differencing of the first derivative of the optimal path gives the second derivative of the optimal path in the form $|D^2(P)| = \sum_t |DP_{t+2} - DP_{t+1}|$ or the Taylor series expansion of the second derivative of the frames gives equally $\sum_t |P_{t+2} - 2P_{t+1} + P_t|$, since we model the error as additive instead of compositional. We therefore minimize directly the difference of the residuals $|R_{t+1} - R_t| = |F_{t+2}B_{t+2} - (I + F_{t+1})B_{t+1} + B_t|$.

3. $|D^3(P)|_1$: Similarly using the Taylor series expansion of the second derivative of the first derivative, $|R_{t+2} - 2R_{t+1} + R_t| = |F_{t+3}B_{t+3} - (I + 2F_{t+2})B_{t+2} + (2I + F_{t+1})B_{t+1} - B_t|$.

4. Minimizing over B_t: As initially mentioned, the known frame-pair transforms F_t and the unknown update transforms B_t are represented by linear motion models. For example, F_t is expressed as a 6 DOF affine transformation $F_t = A(x; p_t) = \begin{pmatrix} a_t & b_t \\ c_t & d_t \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} dx_t \\ dy_t \end{pmatrix}$ with p_t being the parametrization vector $p_t = (dx_t, dy_t, a_t, b_t, c_t, d_t)^T$. Similarly, a 4 DOF linear similarity is obtained by setting $a_t = d_t$ and $b_t = -c_t$.

We seek to minimize the weighted L_1 norm of the residuals overall update transforms B_t parametrized by their corresponding vector p_t. Then, the residual for the constant path segment becomes $|R_t(p)| = |M(F_{t+1})p_{t+1} - p_t|$, where $M(F_{t+1})$ is a linear operation representing the matrix multiplication of $F_{t+1}B_{t+1}$ in parameter form.

Hence, the residuals can be represented as:

$$|D(P)|_1 = \sum_t (|C_t| \cdot |F_{t+1} \cdot B_{t+1} - B_t|)$$
$$|D^2(P)|_1 = \sum_t |R_{t+1} - R_t|$$
$$|D^3(P)|_1 = \sum_t |R_{t+2} - 2 \cdot R_{t+1} + R_t|$$

To solve the optimization problem using the linear programming method, the slack variables e_t^1, e_t^2, and e_t^3 corresponding to each residual are introduced to minimize the L1 norm of the residuals expressed as:

$$-e_t^1 < F_{t+1} \cdot B_{t+1} - B_t < e_t^1$$
$$-e_t^2 < R_{t+1} - R_t < e_t^2$$
$$-e_t^3 < R_{t+2} - 2 \cdot R_{t+1} + R_t < e_t^3$$

where $R_t := F_{t+1}B_{t+1} - B_t$

Then the objective function to minimize can be expressed as:

$$w_1 \cdot c_1^T \cdot e_t^1$$
$$w_2 \cdot c_2^T \cdot e_t^2$$
$$w_3 \cdot c_3^T \cdot e_t^3$$

where c corresponds to $(dx_t, dy_t, a_t, b_t, c_t, d_t)$ (for $|D(P)|_1$).

Adjusting the weights of c enables us to steer the minimization towards specific parameters. For instance, using a weighting of 100:1 for affine to translational parts helps us to suppress the affine transformations more than the translational transformation and reduces the effects of possible deformations that can occur in the video frame.

$$c = [1, 1, 100, 100, 100, 100, 100]$$

F. Inclusion and Proximity Constraints

The inclusion constraint requires the crop window with the specified size to stay contained inside the original camera frame. Hence, this ensures that all pixels in the crop window contain valid values.

The proximity constraint aims to follow the general intent of the original video by limiting the values of scaling and shearing. This constraint also limits the difference between the operations done on the x-axis and y-axis to prevent possible deformations that can occur on the objects in the crop window.

$$F_t = \begin{pmatrix} a_t & b_t & \Delta x_t \\ c_t & d_t & \Delta y_t \\ 0 & 0 & 1 \end{pmatrix}^T \tag{7}$$

The constraints for the parameters in the transformation matrix can be listed as:

$$0.9 \leq a_t \leq 1.1$$
$$0.9 \leq d_t \leq 1.1$$
$$-0.1 \leq b_t \leq 0.1$$
$$-0.1 \leq c_t \leq 0.1$$
$$|a_t - d_t| \leq 0.05$$
$$-0.1 \leq b_t + c_t \leq 0.1$$

These constraints mean restricting the magnitude of the scaling and shearing. Also, the difference between the scaling along the x and y axes is also restricted so that the deformation in the output video can be reduced. Without these constraints, the optimal path would be constant.

G. Stabilizing the Video

After solving the optimization problem, the transform B_t which transforms a crop window originally centered in the frame rectangle is obtained. In order to apply those desired transforms to the shaky frames and "undo" the shaking, the function `warpAffine` is used from the `cv2` library. The function takes the frame along with the transformation matrix and applies the transformation to the frame so that it can be used in the stabilized video.

III. Results

The stabilized videos for different scenarios and the plots for the motion paths along the x and y axes of the stabilized videos are obtained to show and evaluate the results obtained.

As can be seen from the plots below shaky and undesired motion paths are eliminated and instead smoother but still meaningful which contains the intent of the original motion path is obtained.

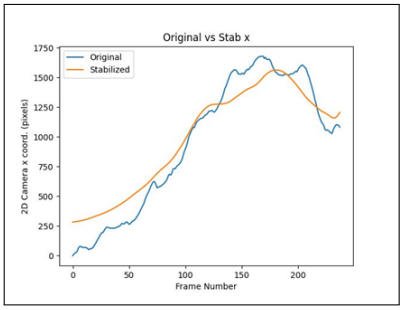

Fig. 4. Stabilized and original motion paths along the x direction.

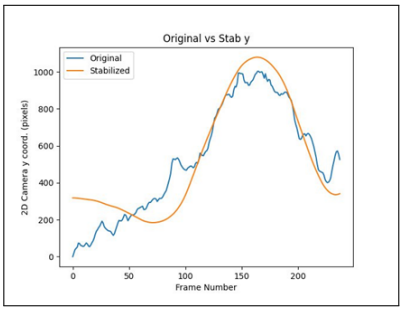

Fig. 5. Stabilized and original motion paths along the y direction.

In the screenshots taken from the original and stabilized video frames, we can observe that the object which is the home contained inside the crop window has a little bit of deformation in the x-y axes. This happens because of the allowed scaling differences in the x-y axes. That implies that the restrictions on the translational and affine parts of the transformations should be put wisely to obtain a desired stabilized video frame.

Another way to avoid deformations on the object of interest is to obtain so many feature points on that specific object. This method was applied to the videos containing faces using the face-tracking algorithm and fruitful and better results were obtained.

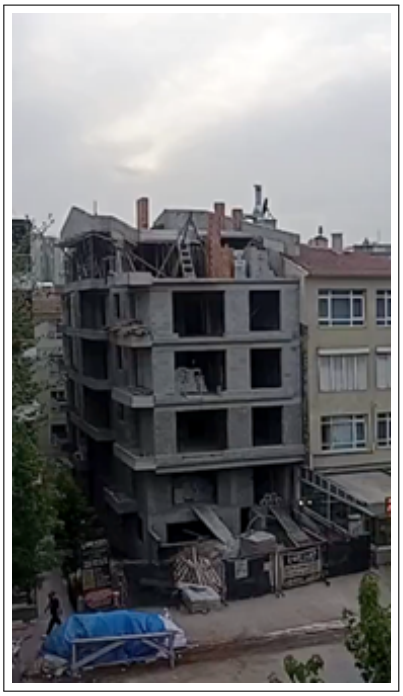

Fig. 6. Original video frame.

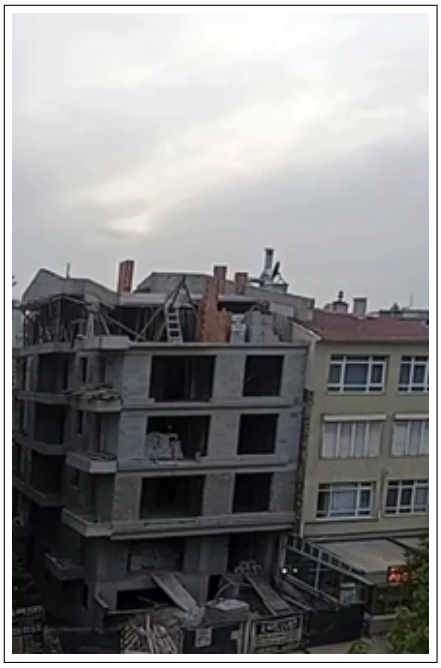

Fig. 7. Stabilized crop window frame.

IV. Conclusion

In conclusion, the approach presented in this paper demonstrates different approaches for the development of a stabilized video system utilizing L1 optimal camera paths. By employing a multi-step methodology outlined in the Auto-Directed Video Stabilization with Robust L1 Optimal Camera Paths paper, we have successfully stabilized shaky videos by estimating and optimizing camera paths.

The implementation process involved several key steps. Initially, we selected high-quality feature points in video frames using the Lucas-Kanade optical flow algorithm, ensuring accurate tracking of motion. Subsequently, we modeled the camera's motion using affine transformations between consecutive frames, ultimately deriving the original camera path C_t. Through the use of linear programming optimization, we minimized the weighted sum of derivatives of the optimal path, effectively smoothing the camera trajectory while preserving the original intent of the motion. Additionally, inclusion and proximity constraints were enforced to ensure the stability and consistency of the final output.

The results of our implementation have been promising, as evidenced by the stabilized videos and motion path plots. The elimination of undesired camera motions and the preservation of meaningful motion patterns signify the effectiveness of our approach. However, it's crucial to note that some deformations may occur, specifically because of the use of scaling and shearing. To mitigate this, careful adjustment of constraints and feature selection strategies can be employed, as demonstrated in our experimentation with face-tracking.

Overall, this paper showcases a robust framework for video stabilization, laying the groundwork for further refinement and optimization. With continued development and fine-tuning,

this approach holds the potential to enhance video quality across various applications, from amateur film-making to professional cinematography.

V. Future Work

The work done in this paper can be extended and improved in the future. We showed that for video stabilization, using carefully selected features significantly improves stabilization, especially for methods based on camera paths and affine transformations.

As it was demonstrated that face tracking improved the stabilization results, further extension of this work can be done by doing tracking on many objects in the video's frame. A possible approach is to track all the "main" objects in the video's frame and then extract features from only those objects, ignoring the background and other trivial features. This will make the stabilization algorithm more robust and better adapted to deal with different scenarios where we might not have any faces in the videos, or the faces are not easily detectable or trackable.

Another approach would be to fix the tracking based on the use case and application area. This will probably lead to better results in settings where it is possible to expect what objects to have, so that the tracking algorithm can focus on only those main objects.

References

[1] M. Grundmann, V. Kwatra, and I. Essa, "Auto-directed video stabilization with robust l1 optimal camera paths," in *CVPR 2011*, Jun. 2011, pp. 225–232. DOI: 10.1109/CVPR.2011.5995525.

[2] A. K., *CV2.GOODFEATURESTOTRACK()*, https://theailearner.com/tag/cv2-goodfeaturestotrack/, [Online; accessed 29-March-2024], 2021.

[3] L. Schmid, *Local Feature Detectors*, [Online; accessed 29-March-2024], 2003.

[4] Joefernandez, *MediaPipe Face Mesh*, https://github.com/google-ai-edge/mediapipe/blob/master/docs/solutions/face_mesh.md, [Online; accessed 16-May-2024].

[5] OpenCV, *Optical Flow*, https://docs.opencv.org/4.x/db/d7f/tutorial_js_lucas_kanade.html, [Online; accessed 29-March-2024].