Final Project Report: Video Stabilization with Classical Methods

Jiuhong Xiao (jx1190) and Jennifer Yeom (jy4037)
Project Mentor: Junbo Chen

***Abstract*—Video stabilization technology is an essential part of our daily lives among applications in various contexts such as shooting videos on a mobile phone or capturing aerial images with a flying drone. This project aims to explore video stabilization through classical methods and develop a framework to perform motion estimation and correction. The experiment is conducted using a publicly available dataset *DeepStab* and our own Unmanned Aerial Vehicle (UAV) video. The results of the study are used to compare the stabilization performance of different motion estimation and correction approaches. We also show an example stabilized video in this [link] from the *DeepStab* dataset, as well as an example stabilized video [link] from a quadrotor UAV flown in our lab space. By doing so, the project provides insights into the effectiveness of classical methods for video stabilization. Our code can be found in this [link].**

I. Overview

Video stabilization is defined as a task to smooth the oscillation and disturbance of the video from a moving camera. For our project in Image and Video Processing, we are interested in video stabilization with classical methods. As students in the Agile Robotics and Perception Lab (ARPL), we work closely with quadrotors, small drone-type helicopters with four rotors. Our research spans the control, perception, navigation, and localization of a quadrotor, all of which can be highly dependent on interpreting video information from a visual sensor. One can imagine the tilt or motion blur of video input to a quadrotor's perception or control algorithm could be negatively impacted. These effects are also highly likely as quadrotors can move quickly, with jerky changes in direction and speeds up to 8-9 m/s. We chose to explore the problem of video stabilization with classical methods as the approach will have low computation complexity and the results more interpretable. Through this project, we evaluate different video stabilization methods on a public dataset and then test the applicability to a quadrotor.

A video stabilization system can be divided into motion estimation and motion correction [1]. The first part is to model the 2D or 3D trajectories of cameras from video frames, and the second is to smooth the trajectories and synthesize the stabilized video. Motion estimation includes initial motion estimation and outlier rejection. Motion correction requires motion smoothing and video synthesis.

To compare the performance between different methods, we implement different types of motion estimation and motion correction schemes. Specifically, we compare optical flow v.s. feature matching methods and affine v.s. homography transformations for motion estimation. In terms of motion correctness, we implement motion smoothing with a low-pass filtering technique. As video stabilization results are difficult to quantify, we analyze our results with qualitative results.

For the datasets, we use the *DeepStab* dataset [2], a public video stabilization dataset with synchronized pairs of steady and unsteady video frames. The videos are captured by hand-holding GoPro Hero 4 cameras. The pairs of raw-stabilized video frames can help us to evaluate the stabilization performance with ground truth. We also use videos recorded from a quadrotor to evaluate our framework.

II. Background

In this section, we describe the various algorithms as the background knowledge for our following implementation.

A. Shi-Tomasi corner detector

The Shi-Tomasi corner detector [3] uses the minimum eigenvalue to detect corners. Given an image $I(x, y)$, the first step of the Shi-Tomasi detector is to compute the gradient image $I_x(x, y)$ and $I_y(x, y)$. The second moment matrix M at each pixel (x, y) is then computed as:

$$M = \begin{bmatrix} \sum wI_x^2 & \sum wI_xI_y \\ \sum wI_xI_y & \sum wI_y^2 \end{bmatrix}, \tag{1}$$

where w is a Gaussian window function that weights the contributions of nearby pixels. The eigenvalues λ_1 and λ_2 of matrix M are then computed, and the minimum eigenvalue $\lambda_{\text{min}} = \min(\lambda_1, \lambda_2)$ is used to compute a corner response function:

$$R = \lambda_{\text{min}}. \tag{2}$$

The corners with the highest values of R are selected as the best features or corners of the image.

B. Lucas-Kanade optical flow

The Lucas-Kanade [4] optical flow algorithm estimates the motion between two consecutive frames, using features from feature detectors such as the Shi-Tomasi corner detector. The algorithm assumes small and local linear motion. The motion vector is computed using the following linear system of equations:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x \Delta I \\ \sum I_y \Delta I \end{bmatrix}, \tag{3}$$

where I_x and I_y are the directional gradients of original image frame I, and $\Delta I = I(x+u(x,y), y+v(x,y), t+1) - I(x,y,t)$ which is the difference between the two frames. The computation is done at each pixel (x, y) but is now shown in the equation as $I_x(x, y)$ for example, for concise notation.

C. ORB feature descriptor

The ORB (Oriented FAST and Rotated BRIEF) [5] algorithm detects key points in a given image using the FAST (Features from Accelerated Segment Test) corner detector. FAST is a corner detector that compares the brightness of a pixel with its neighbors in a circular pattern [6]. Once the corners are detected, the dominant orientation is determined by computing a gradient orientation histogram. Next, the algorithm computes a binary descriptor using the BRIEF (Binary Robust Independent Elementary Features) descriptor [7]. The ORB feature descriptor is both scale and rotation invariant. The ORB descriptors are compared using the Hamming distance, which measures the number of bit differences between the two binary strings. The feature point with the lowest Hamming distance is regarded as the best-matched feature point.

D. Affine transformation

An affine transformation is a linear mapping that preserves parallel lines and ratios of distances between points. It can be represented by $\mathbf{p} = [x, y, 1]^T$ with a 3×3 matrix $\mathbf{A}$:

$$\mathbf{p}' = \mathbf{A}\mathbf{p} \tag{4}$$

where $\mathbf{p}' = [x', y', 1]^T$ is the transformed point. An affine transformation can be generally decomposed into three operations: translation, rotation, scaling, and shearing. However, we do not consider shearing in our case. With only translation, rotation, and scaling, the matrix $\mathbf{A}$ has the following form:

$$\mathbf{A} = \begin{bmatrix} s\cos r & -s\sin r & t_x \\ s\sin r & s\cos r & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

where s is the scaling term, t_x, t_y is the translation coefficients, r is the angle of rotation, and the last row of zeros and ones ensure that the transformation is affine. The affine transformation has 6 degrees of freedom and requires at least 3 sets of points to compute.

E. Homography transformation

A homography transformation is a perspective transformation that maps points in one image to points in another image taken from a different camera view. The homography transformation is a more general solution compared to the affine transformation. It is a 3x3 matrix that describes a projective transformation of a plane onto another plane. The transformation is defined by:

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{6}$$

where (x, y) are the coordinates of the first image, (x', y') are the coordinates of the transformed image, and λ is the scale factor. The homography transformation has 8 degrees of freedom and requires at least 4 sets of matching points to compute.

F. Savitzky-Golay filter

The Savitzky-Golay filter [8] is a low-pass filter that removes noise from a signal using a linear regression method. It fits the signal with a polynomial function with a sliding window of data points. This filter provides smooth motion trajectories while preserving existing peaks that could be important to the motion estimation of a video.

G. RANSAC

The RANSAC (Random Sample Consensus) [9] is an iterative outlier detection method. RANSAC first selects a random subset of data points. This subset is used to estimate the model parameters. The number of data points that fit the model within a tolerance (inliers) is counted. The algorithm repeats this process for a user-specified number of iterations and outputs the model with the highest number of inlier points. We use RANSAC in

line with our motion estimation (particularly in feature selection) for outlier rejection for more robust motion estimation results.

III. Project Accomplishment

In this section, we list the accomplished steps of our project.

A. Dataset Selection and Analysis

For evaluating the video stabilization performance, we search for a potential public dataset for video stabilization. We overview three public datasets and list the property of these datasets in Table I. We found *DeepStab* to be a good starter dataset since the dataset has ground truth stabilized videos and the dataset contains high diversity, low-motion, and clear videos. Therefore, our experiments are mainly conducted on this dataset.

B. Motion Estimation: Optical Flow

As introduced in Section II-B, one of the most popular choices for motion estimation of video stabilization is the Lucas-Kenade method [4] for creating optical flow. This method assumes optical flow between two consecutive frames can be approximated as a constant vector within a local neighborhood of pixels. Based on this assumption, the algorithm estimates the optical flow vector for each pixel in the image using the least squares fit.

We explored a variety of existing implementations and decided to adapt the sparse optical flow motion estimation pipeline in [12], [13], which uses the Object Tracking library from OpenCV. Specifically, we use the Shi-Tomasi method for detecting corners and the Lucas-Kenade method for calculating the optical flow as discussed previously. At this stage, we also assume that the transformation between the previous and current frame is a 2D affine transformation or homography transformation. We use the RANSAC [9] algorithm to remove the outliers, which is implemented inside *cv2.estimateAffinePartial2D* and *cv2.findHomography*. The motion estimation algorithm is described in Algorithm 1. We have tested the functionality of the motion estimation pipeline on our dataset, as seen in Fig. 1.

C. Motion Estimation: Feature matching

Although the calculation of optical flow is simple and fast, the performance is unstable when high motion changes the frame a lot or there are no available corner features. Using feature-matching-based motion estimation becomes a plausible solution for these scenarios.

Algorithm 1 Optical-flow-based Motion Estimation

```
Set parameters for optical flow
while ∃ next frame do
    RGB frame → Grayscale
    Detect features (Shi-Tomasi method)
        cv2.goodFeaturesToTrack
    Calculate optical flow (Lucas-Kenade method)
        cv2.calcOpticalFlowPyrLK
    Choose one of the following:
        1) Estimate 2D affine transformation matrix and calculate rotation r and translation factor t_0, t_1
        cv2.estimateAffinePartial2D
        2) Estimate Homography transformation and calculate H
        cv2.findHomography
end while
```

Figure 1: An example image of the optical flow calculation of a video frame. The green lines on the image are the trajectories of the calculated optical flow.

The feature-matching-based motion estimation uses feature detection and extraction methods like ORB [5] to extract feature points and then do matching between previous and current frames. The matching (K-Nearest Neighbors algorithm, KNN) will search for the whole image, instead of a local neighborhood region in optical flow. Finally, we calculate the 2D affine transformations or homography transformations from the matched feature points. The algorithm is described in Algorithm 2.

We have tested the functionality of the motion estimation pipeline on our dataset, as seen in Fig. 2. We can observe some wrong-matched features and they will be excluded from motion estimation by RANSAC.

We summarize the advantage and disadvantages of optical-flow-based and feature-matching-based motion estimation algorithms. The optical-flow-based method uses simple corner features and only matches a local neighborhood of pixels. The advantage is the low computation complexity during feature extraction and matching

Table I: The overviewed public dataset for video stabilization

Author	Description	Num. of Videos	Frame Resolution	With Ground Truth
Wang et al. [2]	Deep-learning Video Stabilization Dataset (DeepStab)	61	1280x720	Y
Zhang et al. [10]	Full-Reference Stability Assessment Dataset	45	Variable	Y
Yu et al. [11]	Labeled Selfie Videos	33	854x480	N

Algorithm 2 Feature-matching-based Motion Estimation

Set parameters for feature matching
while ∃ next frame **do**
    RGB frame → Grayscale
    Detect features (ORB [5])
        cv2.ORB_create
    Match features (KNN)
        cv2.BFMatcher
    Choose one of the following:
        1) Estimate 2D affine transformation matrix and calculate rotation r and translation factor t_0, t_1
        cv2.estimateAffinePartial2D
        2) Estimate Homography transformation and calculate H
        cv2.findHomography
end while

Figure 2: An example image of the feature matching calculation of a video frame using ORB features. The green lines on the image are matched features.

and the estimated motion is smooth. The disadvantage is that the algorithm will fail when the camera is in high motion. The feature-matching-based method uses ORB features and matches the whole image. The advantage is that the performance is stable when the camera is in high motion. The disadvantage is the high computation complexity of feature extraction and matching and the estimated motion may not be noncontinuous, which is a strong disturbance for the following motion smoothing.

D. Motion correction: Offline motion smoothing

For the motion correction implementation part, we implement an offline pipeline. After getting the sequence of transformations, we use a low-pass filter (Savitzky-Golay filter [8]) to smooth the trajectories. The trajectories are defined as the cumulative sum of all previous transformations. After we get the smoothed trajectories, we reconstruct the transformations and apply them to the frames to generate the new video. In the experiments, we find that the scaling factor of affine transformation is stable so we do not need to smooth them but we still need them to do the affine transformation. The algorithm is described in Algorithm 3. We show the stabilized trajectories in Fig. 3 and 4.

Algorithm 3 Offline Motion Smoothing

Calculate the trajectories from the transformation (*np.cumsum*)
Apply the Savitzky-Golay filter to the trajectories (*scipy.signal.savgol_filter*) and reconstruct the transformations
while ∃ next frame **do**
    Choose one of the following:
        1) Apply smoothed 2D affine transformation matrix to video frames
        cv2.warpAffine
        2) Apply smoothed homography transformation matrix to video frames
        cv2.warpPerspective
end while

IV. Results

To qualitatively analyze the results of video stabilization, we represent the example stabilized videos in this section and compare the stabilization performance between different settings.

1) Comparison between affine and homography transformation: We show the results of both affine transformation [link] and homography transformation [link]. From the results, it is evident that the affine transformation has notable distortion, whereas the homography transformation has no obvious deformation. The reason is that the affine transformation is too simple and cannot

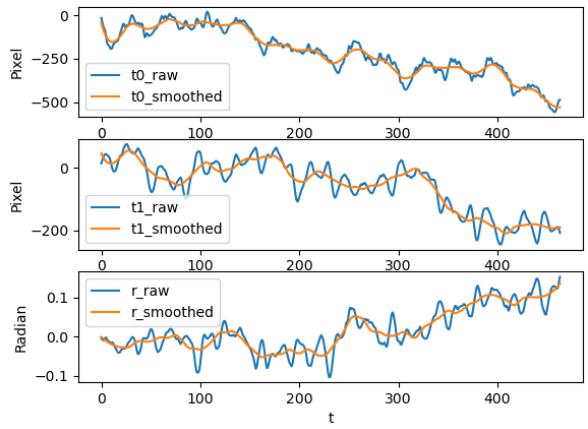

Figure 3: The smoothed trajectories of 2D affine transformation with the translation factors t_0, t_1, and rotation factor r

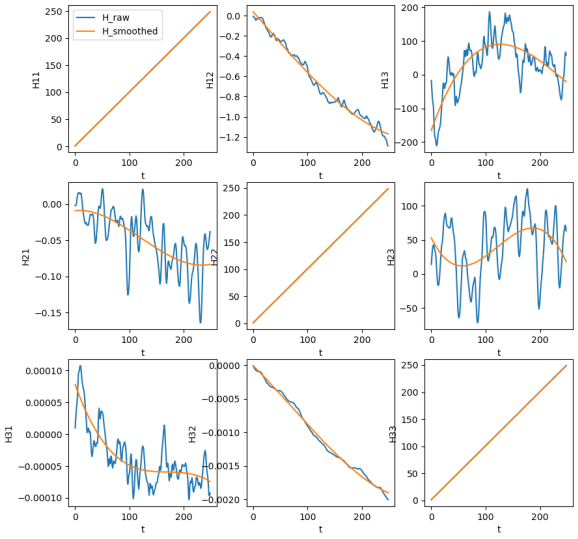

Figure 4: The smoothed trajectories of homography matrix H

correctly reflect the perspective transformation of the images from the actual camera motion.

2) Result of Feature-matching-based method: The result of the feature-matching-based method with ORB features and homography transformation is shown in [link]. From the result, we can observe that the stabilized video lacks smoothness due to the noncontinuous motion estimation from ORB features. For a large motion, we recommend using a feature-matching-based method, as it has the capacity to identify matching points throughout the entire image rather than a nearby region in optical flow. However, the expansion of the search region could potentially result in a higher error rate for feature point matching, which can degrade the stabilization performance. Also, without the constraints of the search region, the motion between the last frame and the current frame can be noncontinuous, which causes the jitters in the video result.

3) Result of the UAV video: We present the result of our video stabilization approach on our UAV video in [link]. The result demonstrates that our video stabilization method can successfully stabilize the shaking video during the take-off, flying, and landing phases of our UAV, especially the last one. In the real-life experiment, we find that there are three factors that can disrupt the motion estimation: 1) Large motion; 2) scenes with small depth, and 3) flat or self-similar structure. Recognizing these limitations is crucial when considering the deployment of video stabilization technology in UAV applications.

V. Summary

To summarize, we implement offline motion estimation and motion correction pipelines using various existing techniques. We use 2D affine transformation or homography transformation to model the motion of the camera. We only estimate the global transformation, meaning moving objects will affect the results. This pipeline is simple and fast while some distortion will display in the image when the camera is strongly shaking.

Overall, optical flow with corner detector methods outperformed the ORB feature matching for the videos we stabilized and evaluated. As the video motion was small from the hand-held GoPro videos, optical flow performed well, whereas ORB had more jittery results for non-smooth estimated motion. The homography transformation outperformed the affine transformation methods. Using the same input videos, the result of the homography had less jitter over the affine transformation.

Future works related to this project, specifically tailored for quadrotor applications, would be to implement the motion estimation and smoothing in real-time on the onboard computer of the quadrotor itself.

References

[1] W. Guilluy, L. Oudre, and A. Beghdadi, "Video stabilization: Overview, challenges and perspectives," *Signal Processing: Image Communication*, vol. 90, p. 116015, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0923596520301697

[2] M. Wang, G.-Y. Yang, J.-K. Lin, S.-H. Zhang, A. Shamir, S.-P. Lu, and S.-M. Hu, "Deep online video stabilization with multi-grid warping transformation learning," *IEEE Transactions on Image Processing*, vol. 28, no. 5, pp. 2283–2292, 2019.

[3] C. Tomasi and T. Kanade, "Detection and tracking of point," *Int J Comput Vis*, vol. 9, pp. 137–154, 1991.
[4] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.
[5] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.
[6] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*. Springer, 2006, pp. 430–443.
[7] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792.
[8] A. Savitzky and M. J. Golay, "Smoothing and differentiation of data by simplified least squares procedures." *Analytical chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.
[9] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, p. 381–395, jun 1981. [Online]. Available: https://doi.org/10.1145/358669.358692
[10] L. Zhang, Q.-Z. Zheng, H.-K. Liu, and H. Huang, "Full-reference stability assessment of digital video stabilization based on riemannian metric," *IEEE Transactions on Image Processing*, vol. 27, no. 12, pp. 6051–6063, 2018.
[11] J. Yu, , and R. Ramamoorthi, "Selfie video stabilization," *ECCV*, 2018.
[12] N. Nielsen, "Computer vision/optical flow," https://github.com/niconielsen32/ComputerVision, 2021.
[13] A. S. Thakur, "Video stabilization using point feature matching in opencv," https://learnopencv.com/video-stabilization-using-point-feature-matching-in-opencv/, 2019.