UNIVERSITY OF CALIFORNIA SAN DIEGO

Robust Selfie and General Video Stabilization

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Communication Theory and Systems)

by

Jiyang Yu

Committee in charge:

Professor Ravi Ramamoorthi, Chair
Professor Nuno Vasconcelos, Co-Chair
Professor Manmohan Chandraker
Professor David Kriegman
Professor Truong Nguyen

2020

The dissertation of Jiyang Yu is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

Chair

University of California San Diego

2020

DEDICATION

To my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

My sincere gratitude goes first and foremost to my advisor, Prof. Ravi Ramamoorthi, for his guidance throughout my Ph.D. career. The transition from coding theory research to computer vision and graphics research was never easy for me. Without Ravi's enormous patience and generous support, the completion of this thesis would not have been possible. The experience of working with him will be an invaluable asset to my life. I would like to extend my gratitude to Prof. Nuno Vasconcelos, Prof. Manmohan Chandraker, Prof. David Kriegman, and Prof. Truong Nguyen for serving on my thesis committee and their helpful comments.

My appreciation also extends to my collaborators during my Ph.D. journey, Dr. Zexiang Xu, Dr. Jannik Boll Nielsen, Matteo Mannino, Prof. Henrik Wann Jensen, Dr. Jie Hu, Dr. Mohammad Gharavi, Dr. Keli Cheng, Dr. Michel Sarkis, Dr. Ning Bi, and others for their great efforts invested in our projects and publications. Most of my works received great help from my labmates, Dr. Zexiang Xu, Zhengqin Li, Muhammad Riaz, Matteo Mannino, Dr. Zachary Murez, Kai-En Lin, Sai Bi, Tiancheng Sun, Alexandr Kuznetsov, Dr. Lifan Wu, and many others. The constructive discussion with them has always been a precious source of research ideas and I have learned a lot from them.

I would like to thank my friends Xiaoming Huang, Huan Hu, Jimeng Zheng, Youbin Mo, Weihao Wang, Tianxiang Shen, Austin Bustos, and many others for their accompanying me through all the difficult moments in my life. Last but not the least, I would like to thank my parents Baogang Yu and Haixia Zang for the unconditional love and courage they gave to me. My gratitude to them is beyond what I can express in words. I would like to dedicate this dissertation to my parents.

The work in this dissertation is supported in part by the Ronald L. Graham Chair, Qualcomm FMA Fellowship, Sony, and the UC San Diego Center for Visual Computing.

Chapter 2 is a reformatted version of the material as it appears in "Selfie Video Stabilization," Jiyang Yu and Ravi Ramamoorthi [YR19b]. The material has been submitted and accepted

to the IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 3 is a reformatted version of the material as it appears in "Real-Time Selfie Video Stabilization," Jiyang Yu, Ravi Ramamoorthi, Keli Cheng, Michel Sarkis and Ning Bi [YRC+20]. The material has been submitted to the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 4 is a reformatted version of the material as it appears in "Robust Video Stabilization by Optimization in CNN Weight Space," Jiyang Yu and Ravi Ramamoorthi [YR19a]. The material has been submitted and accepted to the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2019. The dissertation author was the primary investigator and author of this paper.

Chapter 5 is a reformatted version of the material as it appears in "Learning Video Stabilization Using Optical Flow," Jiyang Yu and Ravi Ramamoorthi [YR20]. The material has been submitted and accepted to the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2020. The dissertation author was the primary investigator and author of this paper.

VITA

2007-2011	B. S. in Electrical Engineering, Southeast University, China
2011-2013	M. S. in Electrical Engineering, Washington State University
2014-2020	Ph. D. in Electrical Engineering (Communication Theory and Systems), University of California San Diego

PUBLICATIONS

Jiyang Yu and Ravi Ramamoorthi, "Learning Video Stabilization Using Optical Flow", in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2020.

Jiyang Yu and Ravi Ramamoorthi, "Robust Video Stabilization by Optimization in CNN Weight Space", in *IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2019.

Jiyang Yu and Ravi Ramamoorthi, "Selfie Video Stabilization", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019, DOI:10.1109/TPAMI.2019.2931897.

Jiyang Yu and Ravi Ramamoorthi, "Selfie Video Stabilization", in *European Conference on Computer Vision*, Sep. 2018.

Zexiang Xu, Jannik Boll Nielsen, Jiyang Yu, Henrik Wann Jensen and Ravi Ramamoorthi, "Minimal BRDF Sampling for Two Shot Near Field Reflectance Acquisition", in *SIGGRAPH Asia*, Dec. 2016.

Jiyang Yu , Zexiang Xu, Matteo Mannino, Henrik Wann Jensen and Ravi Ramamoorthi, "Sparse Sampling for Image Based SVBRDF Acquisition", in *Workshop on Material Appearance Modeling, EGSR*, Jun. 2016

ABSTRACT OF THE DISSERTATION

Robust Selfie and General Video Stabilization

by

Jiyang Yu

Doctor of Philosophy in Electrical Engineering (Communication Theory and Systems)

University of California San Diego, 2020

Professor Ravi Ramamoorthi, Chair

Video stabilization is one of the most widely sought features in video processing. The problem of video stabilization typically consists of two steps: video motion analysis and stabilized frame synthesis. Traditional video motion analysis relies heavily on local image features and is prone to error in the presence of occlusions and motion blur. It is also challenging to avoid visual artifacts in the stabilized frame synthesis due to complexity of the scenes, e.g. parallax, dynamic objects. In this dissertation, we focus on exploring video stabilization methods that generate high quality stable results and are generalizable to real world videos.

We start from selfie videos, which is a special type of video that is challenging for all video stabilization methods due to the dynamic occlusion. Our solution is to analyze motion of

the foreground/background separately and jointly stabilize the motion. Specifically, we seek to use 3D human face model as a prior information of foreground motion. For the background, our approach tracks random pixels using optical flow. We exploit non-linear least squares optimization to stablize both the foreground/background motion. To make the process practical for commercial applications, we also designed an online version of the pipeline using a sliding window scheme. We also exploit deep learning techniques to replace optimization, resulting in orders of magnitude speed improvement compared to the state-of-the-art optimization based approaches.

Our works also generalize to the general video stabilization context. General video occlusion is free-form, therefore a more general scheme is required to appropriately constrain the video stabilization. We proposed two different solutions based on this principle. Both of our approaches use optical flow to analyze the motion and generate a dense warp field for stabilizing input frames. Our first approach seeks to constrain the frame warp field using a global linear transformation. In the second we designed specific metrics based on optical flow to exclude dynamic occlusions and motion boundaries. Our experiments show that our approach is more generalizable and produces both visually and quantatively better results compared to previous video stabilization methods.

Chapter 1

Introduction

Video stabilization is a commonly used feature in both video capturing hardware(e.g. Steadicam, gimbal) and video processing software(e.g. Adobe Premiere, Deshaker). For amateurs, using specific video stabilization equipment is usually difficult and expensive. This makes software video stabilization an important topic in computational photography research. In typical video stabilization methods, video motion is first analyzed by feature tracking or optical flow. The motion is then stabilized using optimization over the entire sequence, solving for 2D image transformations or novel camera views to synthesize the stabilized frame.

However, problems exist in the traditional video stabilization pipeline that prevent these methods from being robust to various real-life videos. First of all, feature detection and tracking and optical flow are prone to error themselves. Traditional video stabilization methods either assume the motion detection is accurate and stabilize the video blindly according to this information, or utilize constraints directly derived from these pre-computed motions(e.g. SfM for 3D feature points, maintain spatial location among feature tracks). This makes the feature tracking/optical flow become a performance overhead for video stabilization. In reality, these methods often produce unexpected artifacts due to the complex nature of videos(motion blur, occlusions, rolling shutter etc.). Second, traditional video stabilization algorithms use full-frame, grid homography

or a dense warp field to synthesize the stabilized frames. This is essentially a difficult trade-off: simple parameterized warping cannot handle non-linear motions like rolling shutter, while a dense warp field easily generates artifacts near motion boundaries due to the lack of constraints. Third, video stabilization requires a significant amount of future motion information for motion smoothing, making it difficult to be applied to real-time applications.

In this dissertation, we aim to solve the above challenges from several aspects. In Chapter 2, we start from selfie videos containing dynamic occlusions that are common limitations of video stabilization methods. We re-design the optimization based pipeline in Chapter 2 into learning-based online video stabilization in Chapter 3, making it pratical to be deployed in real-time applications. In Chapter 4, we generalize our work to general videos and use a linear warping constraint on the warp field used for stabilizing the frames. In Chapter 5, we propose a fully automatic learning-based method that generates a warp field for stabilization directly from optical flow fields computed from input video.

For selfie videos, our goal is to automatically generate stabilized video that has optimal smooth motion in the sense of both foreground and background. In Chapter 2, we propose a novel algorithm for stabilizing selfie videos. The key insight is that non-rigid foreground motion in selfie videos can be analyzed using a 3D face model, and background motion can be analyzed using optical flow. We use second derivative of temporal trajectory of selected pixels as the measure of smoothness. Our algorithm stabilizes selfie videos by minimizing the smoothness measure of the background, regularized by the motion of the foreground. Experiments show that our method outperforms state-of-the-art general video stabilization techniques in selfie videos.

Optimization based selfie video stabilization in Chapter 2 requires future frames and is computationally expensive. Based on the same principle that 3D face model can be used to analyze the foreground motion, we design a deep learning based selfie video stabilization. In Chapter 3, we propose a novel real-time selfie video stabilization method that is completely automatic and runs at 26 fps. We use a 1D linear convolutional network to directly infer the

rigid moving least squares warping which implicitly balances between the global rigidity and local flexibility. Our network structure is specifically designed to stabilize the background and foreground at the same time, while providing optional control of stabilization focus (relative importance of foreground vs. background) to the users. To train our network, we also collect a selfie video dataset with 1005 videos. Compared to our previous offline selfie video method, our approach produces comparable quality with a speed improvement of orders of magnitude.

In Chapter 4, we generalize our focus to general videos. Unlike traditional video stabilization techniques that involve complex motion models, we directly model the appearance change of the frames as the dense optical flow field of consecutive frames. We introduce a new formulation of the video stabilization task based on first principles, which leads to a large non-convex problem that is hard to solve. In this chapter, we propose a novel optimization routine that transfers this problem into the convolutional neural network parameter domain. While we exploit the general benefits of CNNs, including standard gradient-based optimization techniques, our method is a new approach to using CNNs purely as an optimizer rather than learning from data. Our method trains the CNN from scratch on each specific input example, and intentionally overfits the CNN parameters to produce the best result on the input example. By solving the problem in the CNN weight space rather than directly for image pixels, we make it a viable formulation for video stabilization.

Chapter 4 explored the possibility of using a neural network as an optimizer for video stabilization. A natural question to ask is if we can pre-train a network that works for arbitrary videos. In Chapter 5, we propose an automatic learning-based solution that infers the per-pixel warp fields for video stabilization from the optical flow fields of the input video. We also propose a pipeline that uses optical flow principal components for motion inpainting and warp field smoothing, making our method robust to moving objects, occlusion and optical flow inaccuracy, which is challenging for other video stabilization methods. This method gives a $\sim$3x speed improvement compared to the optimization based methods.

Chapter 2

Selfie Video Stabilization

2.1 Introduction

In this chapter, we focus on developing a stabilization method for selfie videos which contain dynamic occlusion, a common limitation of existing video stabilization methods. Selfie video has become one of the major video types thanks to the recent development of social media. However, selfie videos taken by amateurs are usually shaky due to the lack of stabilizing equipment. Recent state-of-the-art works have been developed for general video stabilization tasks and integrated into commercial tools such as Adobe Warp Stabilizer[LGJA09] and the YouTube video stabilizer[GKE11]. However, selfie videos usually have properties that create difficulties for existing methods. We show several example frames from typical selfie videos in Fig. 2.1, in which these properties are demonstrated:

(a) Large non-rigid occlusion from face and body close to the camera;
(b) Selfie videos usually come with strong motion blur/out-of-focus background;
(c) Foreground motion does not coincide with background motion.

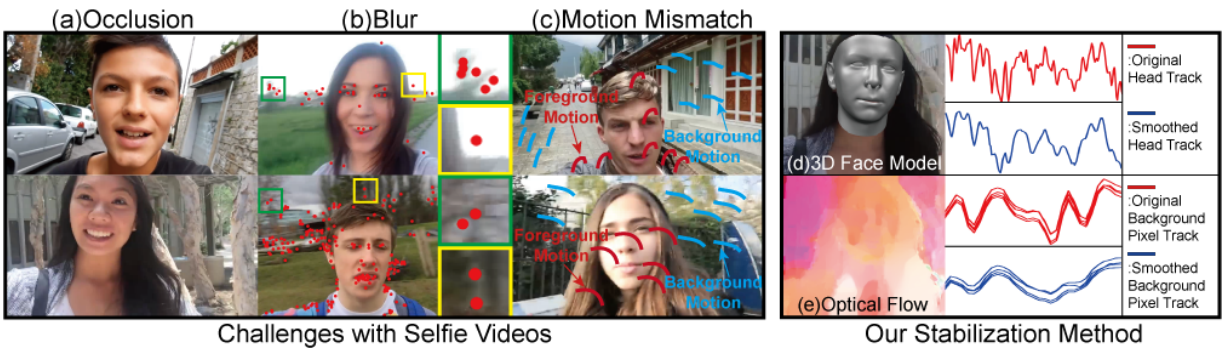


Figure 2.1: *Selfie videos have several properties that cause difficulties for traditional video stabilization methods: (a) face and body significantly occludes the background; (b) bad feature detection caused by motion blur/out of focus, insets show areas where feature points are hard to track accurately; (c) foreground and background motion mismatch, the foreground motion(red) can be different from background motion(blue) due to the dynamics of face and body; Our method uses (d) a 3D face model to analyze the motion in the foreground and (e) optical flow to analyze the motion in the background. The video is stabilized with respect to both foreground and background.*

General video stabilization methods fail in selfie videos for several reasons.

First, most of these works depend on tracking 2D feature points. Existing 3D stabilization approaches require Structure from Motion(SfM) to estimate an initial camera path and build a sparse 3D scene. 2D methods also need to find frame motion using features. Therefore these methods are sensitive to inaccurate tracking of feature points. In Fig. 2.1(b), we show the example frames with blurred background and lack of sharp corners. In these videos, feature point detection is less reliable and the subsequent feature tracking is error-prone.

Second, it is also difficult to obtain long and error-free feature tracks in selfie videos with strong shake. The feature tracking becomes brittle due to the significant occlusion imposed by human face and body. Having noticed feature tracking as a general shortcoming in video stabilization, some methods tried to avoid using features by analyzing the pixel profiles using optical flow[LYTS14]. However, optical flow based algorithms still have failure cases when the occluding object dominates the foreground, which is likely to happen in selfie videos(Fig. 2.1(a)). Our algorithm takes advantage of optical flow to track the background pixels. Unlike Liu et.al[LYTS14] which uses optical flow to synthesize new frames, we only warp the frame with 2D

projective transformations and a grid-based warp field. This guarantees the rigidity over the entire frame. To avoid tracking points and generating long trajectories, we only use small segments of these trajectories so that the foreground occlusion has minimal impact on the stabilization. We further discuss the advantages of the strategy in Sec. 2.7.

Third, general video stabilization only stabilizes with respect to part of the scene. This is not always desired in selfie videos. Both foreground (face and body) and background are important regions that need to be stabilized. To our knowledge, ours is the first method that utilizes the face geometry information in the video stabilization task(Fig. 2.1(d)). Our algorithm can automatically plan the optimal motion so that both the foreground and background motion are smoothed(Fig. 2.1(d)(e)). In summary, our contributions include:

Foreground motion from 3D face model: We utilize 3D human face information to gain knowledge about foreground motion in selfie videos(Sec. 2.4).

Novel background motion tracking: Our method uses optical flow to find dense correspondences on the background, and therefore does not require good feature detection and tracking. We only use temporal motion information and are robust to occlusions in the scene(Sec. 2.5).

Optimal foreground/background stabilization: By considering foreground motion, our method can stabilize selfie videos with respect to foreground and background simultaneously(Sec. 2.6).

Labeled selfie video dataset: We provide a selfie video dataset (Fig. 2.7) of 33 videos, labeled with properties such as dynamic occlusion and lack of background features (Fig. 2.9) that significantly affect the video stabilization task. The dataset can be used to compare different methods, and will be a useful resource for the field. We make the dataset, code and benchmark per Fig. 2.9 publicly available online at `http://cseweb.ucsd.edu/~viscomp/projects/ECCV18VideoStab`.

2.2 Related Work

General video stabilization can be broadly categorized into 2D methods and 3D methods, according to their proposed camera motion models.

2D Stabilization General 2D video stabilization techniques compute 2D motion and generate stabilized video by applying the smoothed motion to original video frames. Some approaches use simple camera motion models. Grundmann et al.[GKE11] proposed a constrainable L1-optimization framework which solves the smoothed camera path composed of constant, linear and parabolic motion. Gleicher and Liu[GL08] assume the scene is largely planar and use homography to synthesize new frames. Liu et al.[LYTS13] divide the frame space into a grid mesh and allow spatially-variant motion. Some methods smooth the motion by imposing non-trivial constraints. Liu et al.[LGW+11] smooth 2D feature tracks by enforcing low-dimensional subspace constraints. Wang et al.[WLHL13] smoothes feature tracks while maintaining the spatial relations among them. Goldstein and Fattal[GF12] uses epipolar constraints when warping the video frames into synthesized frames. There are also explorations of non feature-point based approaches: Liu et al.[LYTS14] solves for a smooth per-frame optical flow field and stabilizes the video by smoothing pixel profiles instead of smoothing feature tracks.

3D Stabilization Some methods sparsely reconstruct the 3D scene. The sparse 3D information is used to guide the synthesis of new frames. These methods generate better results than 2D methods by modeling physically accurate 3D camera motions but are less robust under non-ideal conditions, e.g. large occlusion, motion blur, and rolling shutter. Liu et al.[LGJA09] first uses structure from motion to find feature points' 3D positions, reprojects them onto the smoothed camera path and warps the original frames according to reprojected feature points. There are also methods that render new frames using 3D information: Buehler et al.[BBM01] uses image-based rendering to synthesize new views; Smith et al.[SZJA09] utilize the light field camera to stabilize video; Sun[Sun12] uses depth information. Due to the non-rigid occlusion in

selfie videos(Fig. 2.1(a)), 3D methods that are based on structure from motion can be error-prone. 3D methods that use depth information are also not directly applicable to selfie videos, since depth is not available in most cases.

Face Modeling Human face modeling has been intensely studied. We will only summarize works that are closely related to our work. A widely used early work(Blanz and Vetter[BV99]) models face shape across people as a parametric PCA space learned from a database of laser scans. Cao et al.[CWZ+14] models faces by assembling Kinect face scans as a tensor with identity and expression dimensions. Many follow-up works apply these models in image manipulation (Cao et al.[CWZ+14], Fried et al.[FSGF16]), image/video re-animation(Blanz et al.[BBPV03], Thies et al.[TZS+16]), face tracking(Cao et al.[CHZ14]), facial performance capture(Cao et al.[CBZB15], Shi and Tomasi[SWTC14]), face reconstruction and rigging(Garrido et al.[GZC+16], Ichim et al.[IBP15]). However, these works mainly focus on images/videos captured under ideal conditions(still or stable camera). Our work explores the possibility of utilizing 3D face models in the analysis of selfie videos captured with camera shake. We blend the models in Blanz and Vetter [BV99] and Cao et al. [CWZ+14], to use as the reference model for the face fitting process, which will be discussed in Sec. 2.4.

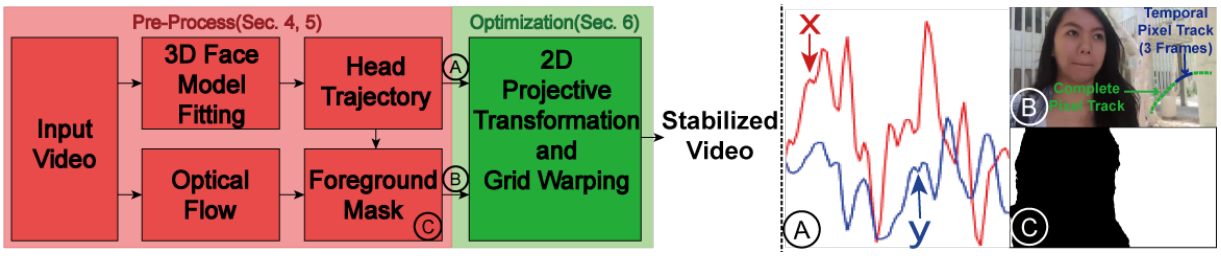


Figure 2.2: *Pipeline of our method. Ⓐ: By fitting a 3D face model, we find the head trajectory in the selfie video(Sec. 2.4); Ⓑ: Optical flow is used to track background pixels for 3 neighboring frames; Ⓒ: The foreground mask is computed from the head trajectory and is used to find the background pixels(Sec. 2.5). The 2D projective transformation and a grid-based warp field is estimated to remove the undesired motion of both foreground and background(Sec. 2.6).*

2.3 Overview

In this section, we provide an overview of our approach for stabilizing selfie videos (Fig. 2.2). We seek to stabilize the selfie video with respect to both foreground and background. We analyze the foreground motion by modeling the face using a 3D face model, and analyze the background motion by optical flow. Fitting the 3D face model to selfie videos provides the head trajectory. We transform each frame according to the head positions so that the foreground regions are roughly aligned across the entire video. Since the foreground regions are aligned, the accumulated motion in this region will be smaller than background regions. Therefore the foreground and background regions can be separated. Details regarding this process will be discussed in Sec. 2.4. The background is defined as the white region in the foreground mask shown in Fig. 2.2. We randomly select pixels in the background that satisfy certain conditions, and use the optical flow to track their motion. Because of occlusion, our method only tracks pixels for 3 neighboring frames. We discuss details of pixel selection and tracking in Sec. 2.5.

The goal of video stabilization is to warp the original video frame so that the undesired frame motions are cancelled. We model the frame motion as a combination of global motion and local motion. The global motion refers to the 2D projective transformation of a frame. Since the frame content is the result of multiple factors, e.g., camera projection, camera distortion, rolling shutter and the 3D structure of the scene, simple 2D projective transformation cannot represent the camera motion accurately. Therefore, we use local motion to refer to any residual motion. Motivated by this analysis, we design our stabilization algorithm as a single joint optimization that simutaneously stabilizes foreground head motion and the background's global and local motion. We will describe details of our joint optimization algorithm in Sec. 2.6.

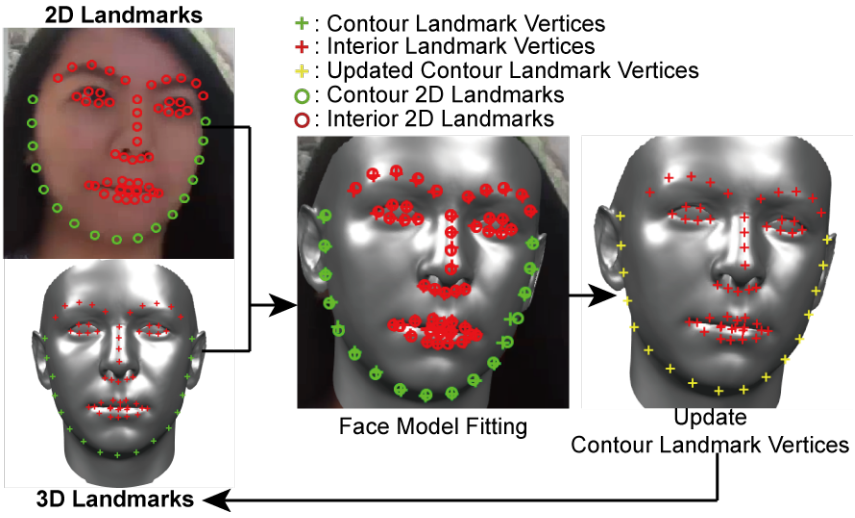


Figure 2.3: *The vertices used as contour 3D landmarks are fixed in the fitting process. The fitted face is rendered and new contour 2D landmarks are detected. The projected vertices closest to the detected 2D contour landmarks are selected as 3D contour landmarks for the next iteration.*

2.4 Foreground Tracking

Since the head and body are attached, we believe that the head motion can well represent the entire foreground motion. We don't explicitly track the body in this work, but implicitly separate the foreground and background by accumulating the optical flow. Details will be discussed in Sec. 2.5. Here we seek to find the position of the head in each frame. Since multiple faces could exist in the selfie video, we only track the dominant face in the video. A regular 2D face detector can provide the face bounding box for each frame, but is not accurate enough for tracking the exact head position. A 2D facial landmark detector provides more accurate detection of the face, but is easily affected by head rotation and facial expression. To find the actual head position invariant to head rotation and facial expression, we use the 3D position of the head and reproject it to the image space as the head position. This requires modeling the face and gaining knowledge about the shape of the face in the selfie video.

3D Face Model We utilize the linear face model proposed by Blanz and Vetter[BV99] and the bilinear face model proposed by Cao et al.[CWZ+14]. Note that although the bilinear face model is more widely used in recent researches, their model was built based on a head mesh

with relatively sparse vertices compared to Blanz and Vetter[BV99]. Our facial landmark based algorithm, which we will discuss later in this section, needs a dense face mesh in the face fitting algorithm. Therefore, we extend the linear face model of Blanz and Vetter[BV99] by transferring the expressions from Cao et al.[CWZ+14]. Our extended linear face model is parameterized as follows:

$$\boldsymbol{F} = \boldsymbol{\mu} + \boldsymbol{U}_s\boldsymbol{\Sigma}_s\boldsymbol{c}_s + \boldsymbol{U}_e\boldsymbol{\Sigma}_e\boldsymbol{c}_e \tag{2.1}$$

where $\boldsymbol{\mu}$ encodes the vertex position of the mean face, $\boldsymbol{U}_s$ are the principal components of face shape, diagonal matrix $\boldsymbol{\Sigma}_s$ contains standard deviations of principal components and $\boldsymbol{c}_s$ is the weight vector that combines principal components. In (2.1), the third term $\boldsymbol{U}_e$ represents the expression principal components. It is generated as follows: we average the shape dimension of the bilinear face model[CWZ+14] and use deformation transfer[SP04] to deform the mean linear face model with the bilinear face model's expressions. We extract principal components $\boldsymbol{U}_e$ of these expression deformed face meshes using regular PCA.

Face Fitting Algorithm Our face model fitting algorithm is a purely landmark based algorithm. For a video with T frames, we detect facial landmarks L_t in each frame using Bulat and Tzimiropoulos[BT17]. The unknown parameters include the 3D rotation $\boldsymbol{R}_t \in SO(3)$, the 3D translation $\boldsymbol{T}_t \in \mathbb{R}^3$, per-frame facial expression coefficient $\boldsymbol{c}_{e,t}$ and the shape parameter $\boldsymbol{c}_s$. We also assume a simple perspective camera projection:

$$\boldsymbol{P} = \begin{bmatrix} f & 0 & w/2 \\ 0 & f & h/2 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where we assume same unknown focal length in horizontal and vertical direction, known fixed optical center at the center of the frame(w and h represents frame width and height respectively), and zero skew. Denoting the 3D transformation matrix as $\boldsymbol{K}_t = [\boldsymbol{R}_t\ \boldsymbol{T}_t]$ where $\boldsymbol{R}_t \in \mathbb{R}^{3\times3}$ and

$\boldsymbol{T}_t \in \mathbb{R}^3$, the 3D face model is fitted by solving:

$$\min_{\substack{\boldsymbol{P},\boldsymbol{R}_t,\boldsymbol{T}_t,\\ \boldsymbol{c}_s,\boldsymbol{c}_{e,t}}} \sum_{t=0}^{T-1} \left(\left\| L_t - \boldsymbol{P}\boldsymbol{K}_t\widehat{\boldsymbol{F}}_t \right\|^2 + \lambda_1 \left\| \boldsymbol{c}_{e,t} \right\|^2 \right) + \lambda_2 T \left\| \boldsymbol{c}_s \right\|^2 \tag{2.3}$$

where $\widehat{\boldsymbol{F}}_t$ represents the landmark vertices controlled by $\boldsymbol{c}_s$ and $\boldsymbol{c}_{e,t}$ as in (2.1), and λ_1 and λ_2 are regularization values that prevent the optimization from getting in local minima. The optimization can be easily solved as an unconstrained non-linear least squares problem. We use all the 199 shape principal components and 46 expression principal components in the fitting process. We use $\lambda_1 = 5$ and $\lambda_2 = 5$ in our experiment. The centroids of fitted face meshes are projected using the solved projection matrix $\boldsymbol{P}$, resulting in the head trajectory.

Facial landmark update In Bulat and Tzimiropoulos[BT17], the contour 2D landmarks are defined by the face silhouette. The face silhouette depends on the pose of the head; therefore the corresponding contour landmark vertices need to be updated during optimization (2.3). However, this requires computing and rendering the whole mesh for facial landmark detection. To avoid this cost, we only update the contour landmark vertices between two iterations of optimization: we first fix the landmark vertices and use them in the face model fitting, then fix the estimated parameters and update the contour landmark vertices. The update of landmark vertices is demonstrated in Fig. 2.3. We first render the current face mesh, and detect 2D landmarks using the rendered image. We update the landmark vertices' indices by projecting all the visible vertices to the image plane and find the closest ones to the detected 2D landmarks. These closest vertices are used as contour landmark vertices in the next iteration. Note that the 2D-3D correspondence is established by finding vertices closest to landmarks. Therefore, a denser mesh will result in more accurate correspondence. This explains why we extend the denser linear face model(Blanz and Vetter[BV99]) instead of using the sparse bilinear face model(Cao et al.[CWZ+14]) directly.

Disscussion General video stabilization methods have difficulties when occlusion occurs

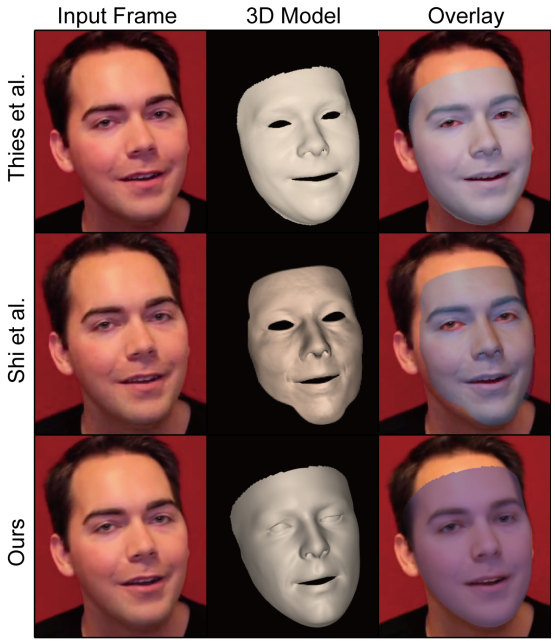


Figure 2.4: *Comparison of our 3D face fitting result to Shi et al. [SWTC14] and Thies et al. [TZS+16]. Our method achieves comparable results without using complex structure-from-motion and shading constraints.*

in the video. Some methods either try to exclude the occlusion region by detecting discontinuity in motions(Liu et al.[LYTS14]) or let users remove features belonging to the foreground(Bai et al. [BAAR14]). The novelty of our method is that it also considers the motion in the foreground. Due to the dynamics of faces, feature based analysis is easily affected by head poses and facial expressions. We use the 3D face model to track the foreground face, so that the foreground can be analyzed even with large non-rigidity. In Fig. 2.4 we show that by implementing the contour landmark update scheme, our face fitting algorithm also achieves results comparable to the methods that use 3D facial landmarks estimated using non-rigid structure-from-motion(Shi et al. [SWTC14]) or 2D facial landmarks with additional light and shading constraints(Thies et al. [TZS+16]). Note that our method uses only 2D landmarks and thus is simpler than state-of-the-art methods.

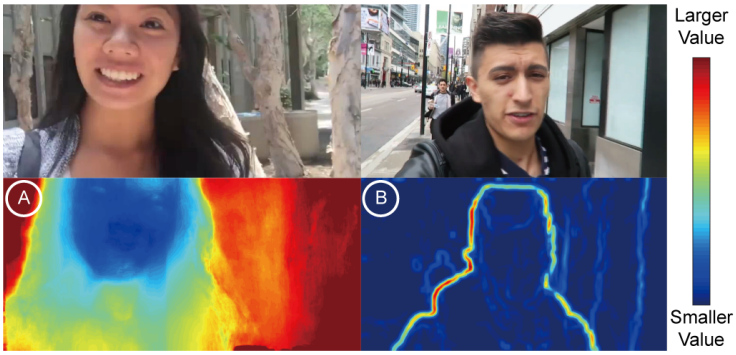


Figure 2.5: Ⓐ*: Accumulated optical flow. A large value indicates the background area.* Ⓑ*: Example moving standard deviation of optical flow. Large values indicate the edges of objects in the scene.*

2.5 Background Tracking

While we can track the foreground motion using a 3D face model, we also need to analyze the background motion so that both these regions can be considered in the stabilization process. We use the optical flow proposed by Kroeger et al. [KTDG16] to track a group of background pixels in each frame. The optical flow can be inaccurate in specific regions due to motion blur/out-of-focus and occlusion. However, minor inaccuracies in small regions can be ignored since our goal is to analyze the global camera motion. In addition, to minimize the impact of occlusion in the scene, we only track each pixel for 3 neighboring frames. We will discuss how this temporal motion information is used in our stabilization process in Sec. 2.6.

Not all pixels can be used to track the background motion. Obviously, pixels falling in the foreground region should not be selected. Face fitting described in Sec. 2.4 provides the head positions in each frame. We first translate all the frames so that the head positions in each frame are aligned to the same point, which leads to a head-aligned video. We perform optical flow between each frame of the head-aligned video. The accumulated optical flow forms a map that encodes the accumulated motion magnitude of each pixel. Since the video is aligned with respect to head position, the accumulated magnitude of optical flow will be smaller in the face and body region, but larger in the background region.

We show an example of a motion map in Fig. 2.5A. After computing the motion map, we use K-means to divide the pixels into two clusters. The cluster with smaller values is considered as foreground. The randomly selected pixels in this cluster will not be used in the stabilization.

Moreover, pixels near the occluding boundary should not be selected. Although our method does not require long feature tracks, we still need to track pixels using optical flow. The downside of tracking with optical flow is that tracking loss caused by occlusion is not easily detectable. To tackle this problem, we want to remove the pixels that are near the occluding boundary.

The objects in the scene can be distinguished by the direction of their motions. We use the standard deviation of the optical flow σ_F in a 21×21 neighborhood to measure the local variation in the optical flow. An example of standard deviation of the optical flow is shown in Fig. 2.5B. The foreground boundary has a larger variation in terms of the optical flow direction. In the stabilization, we only use the pixels with σ_F smaller than a threshold value. We use 0.3 as the threshold in all of our tests.

2.6 Stabilization

The goal of video stabilization is to warp the original video frame so that the undesired frame motions are cancelled. We model the frame motion as a combination of global motion and local motion. The global motion refers to the 2D projective transformation of a frame. Since the frame content is the result of multiple factors, e.g. camera projection, camera distortion, rolling shutter and the 3D structure of the scene, a simple 2D projective transformation cannot represent the camera motion accurately. Therefore, we use local motion to refer to any residual motion.

Motivated by this analysis, we design our algorithm to stabilize the global motion using the whole frame 2D projective transformation and stabilize the local motion using the per-frame grid warping.

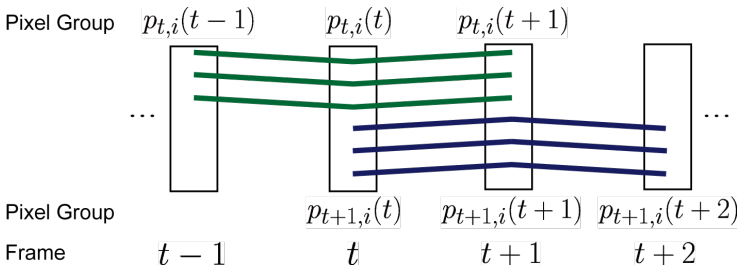


Figure 2.6: *Our method tracks background pixels for 3 neighboring frames.*

Smoothness Measure In selfie videos, the human appears as a large occlusion near the camera, making the trajectory of a pixel fragile. As a consequence, obtaining long feature tracks is difficult. Instead of tracking a pixel over multiple frames, we only track a pixel for 3 frames that are necessary for estimating the second derivative at time t. To demonstrate our idea, we use a single pixel in the scene as an example. Assume a pixel p is tracked over a time period. The trajectory it forms is denoted by $p(t)$. To evaluate the smoothness of this trajectory, we use the integral of squared second derivative or acceleration over the time period. This metric is commonly used in cubic spline fitting algorithms for optimal smoothness. By using this metric, we allow the frames to move to some extent but not try to completely eliminate the low frequency shake. This also helps in generating a larger output frame size when the camera motion is large, which is very common in selfie videos. Details of this effect will be discussed in Sec. 2.7. For a set of selected background pixels (which pixels we choose for this purpose is discussed in Sec. 2.5), the smoothness of the background motion can be written as:

$$E_s(t) = \sum_{i=1}^{N_t} \|\widehat{p}_{t,i}(t+1) - 2\widehat{p}_{t,i}(t) + \widehat{p}_{t,i}(t-1)\|^2 \tag{2.4}$$

where $p_{t,i}$ is the i^{th} pixel tracked from $t-1$ to $t+1$, and $\widehat{p}$ is the new trajectory formed by transforming the original trajectory p. To guarantee the robustness, we track N_t pixels that are randomly selected in the frame at time $t-1$. We illustrate the tracking of background pixels in Fig. 2.6.

Frame Transformation We seek to find a per-frame 2D projective transformation along

with a per-frame grid warp field to transform $p_{t,i}$ to $\widehat{p}_{t,i}$ so that the objective (2.4) is minimized.

For the grid warping, we use the same bilinear interpolation representation as Liu et al.[LGJA09]. Each point is represented by a combination of four vertices of its enclosing grid cell:

$$p_{t,i} = w_{t,i}^T V_t \tag{2.5}$$

where V_t is a vector of the four vertices of the original grid cell that $p(t)$ is in; and w_t is the weight which sums to 1. Denote the output grid as $\widehat{V}$ and the 2D projective transformation as H. The warped scene point $\widehat{p}$ can be calculated using the same weights:

$$\widehat{p}_{t,i} = w_{t,i}^T H_t \widehat{V}_t \tag{2.6}$$

Regularization In selfie videos, the foreground that contains face and body should also be stabilized. The motion of the foreground is not always consistent with the motion of the background. To account for the foreground motion, we also consider the head trajectory:

$$E_h(t) = N_t \left\| \widehat{h}(t+1) - \widehat{h}(t) \right\|^2 \tag{2.7}$$

where $h(t)$ is the head trajectory and $\widehat{h}(t)$ is the transformed head trajectory at time t. The head trajectory was obtained via fitting a 3D face model to the video as described in Sec. 2.4.

Moreover, to avoid undesired deformation caused by grid warping, we use the Laplacian of the grid to measure the rigidity of the warping:

$$E_V(t) = \Delta(\widehat{V}_t) \tag{2.8}$$

Optimization Our final objective function is a combination of the smoothness measure

Figure 2.7: *Example video stills from our dataset. The labels represent the example indices in Fig. 2.9.*

and the regularization term:

$$\min_{H_t,\widehat{V}_t} \quad \sum_{t=1}^{T-2} E_s(t) + \lambda_a E_h(t) + \lambda_b E_V(t) \tag{2.9}$$

Due to the high degree of freedom of the unknowns, the objective function has a complex landscape. Therefore, we first fix the grid $\widehat{V}_t$ and solve for 2D perspective transformation H_t, then use the result as an initialization and refine by running the full optimization (2.9).

We use Matlab's nonlinear least-squares solver to solve this optimization problem. For each frame at time t, the error terms E_s, E_h and E_V are only affected by 3 frames at $t-1$, t and $t+1$. This leads to a sparse jacobian matrix. Therefore this problem can be efficiently solved.

2.7 Results

In this section, we show example frames of selfie video stabilization, along with the visual comparison of the input video, our result, Liu et al.[LGJA09] and Grundmann et al.[GKE11]. We also show that our method achieves better quantitative results than the comparison methods in both selfie video cases and general video cases. Finally we discuss the advantages of our method over general video stabilization methods. Our results are generated with fixed parameters $\lambda_a = 1$ and $\lambda_b = 5$. On average, our Matlab code takes 15 min in all: 3 min for head fitting, 1 min for optical flow, 8 min for optimization and 3 min for warping and rendering the video on a desktop

Figure 2.8: *Visual comparison of input video, our result, Grundmann et al.[GKE11] result and Liu et al.[LGJA09] result. The frames are scaled by the same factor. Our method generates results with larger frame size and does not introduce distortion. We recommend readers to watch the accompanying video for more visual comparison. Labels represent example id in Fig. 2.9.*

computer with an Intel i7-5930K CPU@ 3.5GHz. We did not focus on speed in this work and we believe that our optimization can be implemented on the GPU in future.

Test Set We collected 33 selfie video clips from the Internet, which is the first such dataset of selfie videos. A subset of our example clips are shown in Fig. 2.7. We label each video with properties that affect video stabilization: dynamic occlusion, multiple faces, large foreground motion, lack of background features, dynamic background and motion/defocus blur (Fig. 2.9). Our test set is available at `http://cseweb.ucsd.edu/~viscomp/projects/ECCV18VideoStab` for comparison of different methods, and we believe it will be a useful resource for the community.

Visual Comparison In Fig. 2.8, the video stills are scaled by the same factor so that their sizes can be compared. Our results have a larger field of view compared to Liu et al.[LGJA09] and Grundmann et al.[GKE11], which is often desired in stabilizing selfie videos. This is because the movement of the camera is large in these examples. The methods proposed by Liu et al.[LGJA09] and Grundmann et al.[GKE11] over-stabilize the background, resulting in a small overlap region among frames. To obtain a rectangular video, most of the regions have to be cropped. Our method considers the foreground and background motion together and allows the frame to move in a low frequency sense. Therefore we avoid over-stabilization with respect to either foreground or

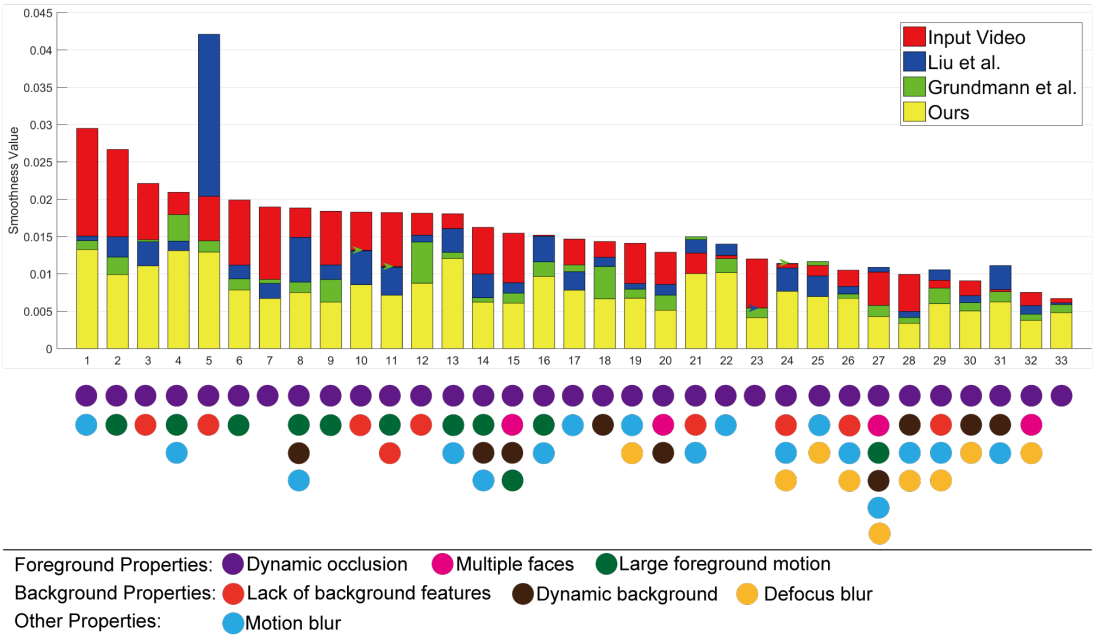


Figure 2.9: *Smoothness comparison of input video, our result, Liu et al. result[LGJA09] and Grundmann et al. result[GKE11]. The horizontal axis represents the examples, and the height of the bar represents the smoothness value. Colored arrows are added where the bars overlap. The labeled properties are visualized as colored dots below each example.*

background. Also note that our result preserves the original shape of the face and body, while the Liu et al.[LGJA09] result contains large distortions on the face. Since the dynamics are hard to show with images, we recommend readers to watch the accompanying video for the visual comparison of the results.

Our method is not sensitive to λ_b, but by changing the head regularization value λ_a in (2.9), we can control the algorithm to mainly stabilize the foreground or the background. We also included an example stabilized with different λ_a values in the accompanying video.

Quantitative Comparision To evaluate the level of smoothness of the videos, we compute the average squared magnitude of second derivative of tracks of all pixels in each frame. The smoothness measure is defined as:

$$S = \frac{1}{|\Omega|} \sum_{t=1}^{T-2} \sum_{i} \|\omega_{t,i}(t+1) - 2\omega_{t,i}(t) + \omega_{t,i}(t-1)\|^2 \tag{2.10}$$

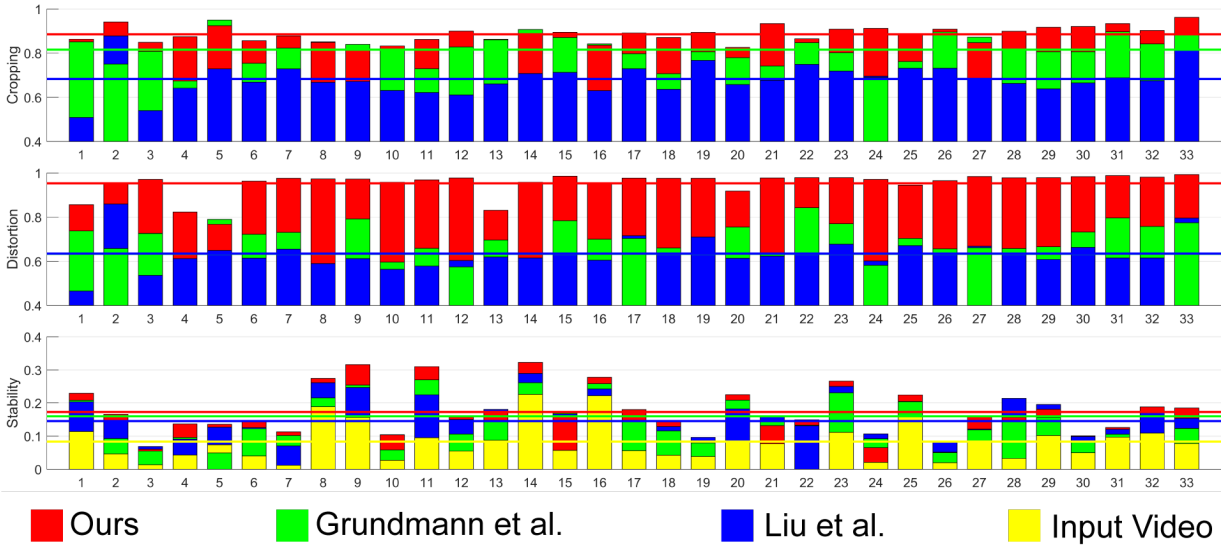


Figure 2.10: *Comparison using 3 metrics suggested in Liu et al.[LYTS13]. Horizontal lines represent the average metric values of the corresponding methods. Note that a higher bar indicates a better result in these metrics.*

where we track all the pixels $\boldsymbol{\omega}_t = \{\omega_{t,1}, \omega_{t,2}...\}$ in the frame at time t, and Ω is the set of all the pixels $\{\boldsymbol{\omega}_1, \boldsymbol{\omega}_2...\boldsymbol{\omega}_{T-1}\}$. Since we sum the second derivatives of all the pixel tracks, a smaller smoothness measure indicates that the frames are changing in a more stabilized way. In (2.10), we use the optical flow to track the pixels. To eliminate the effect of different video sizes, we normalize the optical flow with the frame size on horizontal and vertical directions respectively. We show smoothness comparison for these examples in Fig. 2.9. Note that a lower bar indicates a better result. For better comparison, we sorted the examples by their original smoothness value. Our final results achieve better smoothness compared to the results of Liu et al.[LGJA09] and Grundmann et al.[GKE11] in all of the examples.

For more comprehensive comparison, we also provide quantative comparison with Liu et al.[LGJA09] and Grundmann et al.[GKE11] using the three metrics in Liu et al.[LYTS13]. The result is shown in Fig. 2.10. Note that in these metrics, a higher bar indicates a better result. The colored horizontal lines indicate the average metric values of the corresponding methods over the entire dataset. These results clearly show that our method still performs better under these metrics in the sense of the amount of cropping, distortion and stability. This indicates the effectiveness of

our methods in stabilizing selfie videos.

Advantages Our method has some advantages over other general video stabilization methods in the selfie video case. Traditional 2D and 3D methods usually rely on feature tracks[LGJA09, GKE11, LGW+11, SZJA09], making them vulnerable to insufficient feature counts in selfie videos. Since our method uses optical flow to track the motion, we achieve significantly better result in videos with few background features (examples 3, 5, 10, 11, 12, 21, 24, 26 and 29 in Fig. 2.9). Note that the feature point based general video stabilization methods fail in some of the low feature count cases (examples 5, 21 and 29 in Fig. 2.9), resulting in an even higher smoothness value than the input video. Our method is also robust to videos with motion blur and defocus blur, which are very common properties in selfie videos.

It is hard to obtain long feature tracks in selfie videos with large foreground motion. Note that 3D methods like Liu et al.[LGJA09] cannot perform accurate structure from motion when there is dynamic occlusion. Therefore Liu et al.[LGJA09] in general does not perform well in large foreground motion cases (examples 2, 4, 6, 8, 9, 11, 13, 14, 15, 16 and 27 in Fig. 2.9). Using only fragments of pixel trajectories over 3 frames, our method is robust to large occlusions near the camera. This strategy also helps handle dynamic background (examples 8, 14, 15, 18, 20, 27, 28, 30 and 31 in which multiple non-dominant faces or moving objects exist).

To stabilize the foreground, we need to find the component of head motion that can represent the entire foreground's motion. Our method benefits from a 3D head model since we effectively rule out the head motion purely from rotation and expression. To show this benefit, we compare smoothness measures of example 1, 2, 3, 4, 6, 11 using the 3D head model, the 2D facial landmarks/3D facial landmarks from Bulat and Tzimiropoulos[BT17] and the face bounding box from Viola et al.[VJ01] in Fig. 2.11. In these experiments, we use the centroid of landmarks or the center of the face bounding box as the head position. Note that examples 1, 2, 3 are chosen as general representative examples while 4, 6, 11 are videos with relatively large foreground area or large face rotation/expression. As shown in Fig. 2.11, using 3D head model fitting in general

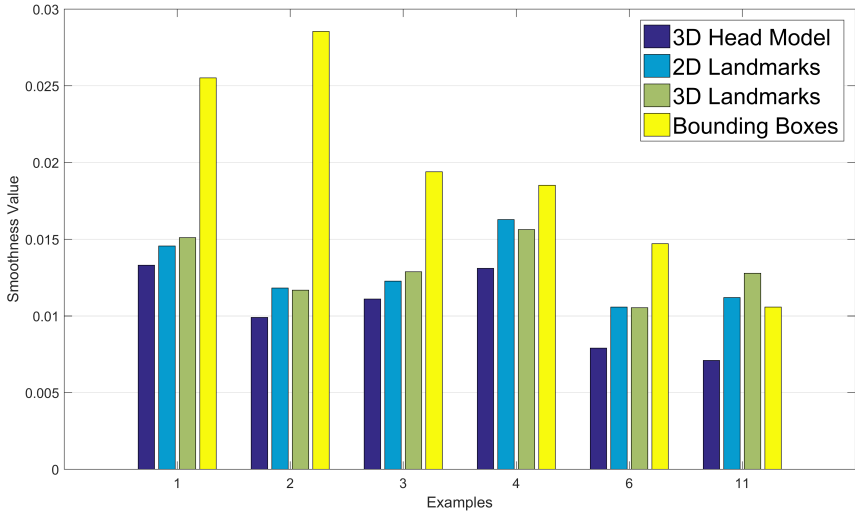


Figure 2.11: *Smoothness comparison using 3D head model, 2D/3D landmarks[BT17] and face bounding box from Viola et al.[VJ01] A lower bar indicates a better result.*

performs better than using just the 2D/3D landmarks or the bounding boxes. In examples 4, 6 and 11, the benefit becomes larger since the 2D/3D landmarks or the bounding boxes are sensitive to head rotation and facial expression change.

Finally, our method provides a novel application of 3D face modeling: track the foreground motion in selfie videos. Current 2D video stabilization methods focus on detecting non-rigid regions and do not consider the motion in these regions. In selfie videos, the foreground occupies a large portion of the frames and cannot be ignored. Our method automatically plans the motion so that both foreground and background motion are smoothed. The foreground motion also helps regularize the video stabilization process. In all of the examples, our method avoids over stabilizing the background and produces results with significantly larger field of view.

Generalization Our method also applies to stabilizing general videos. We can simply ignore the E_h term in (2.9) and perform the optimization for the entire background region. We also collect 6 general videos along with 10 videos from Liu et al.[LGJA09], Grundmann et al.[GKE11] and Liu et al.[LYTS13] shown in Fig. 2.12 and compare the smoothness of our result against the comparison methods. Note that we only use 3 neighbouring frames to track the frame motion and only local motion information is available. Therefore, our method faces a harder

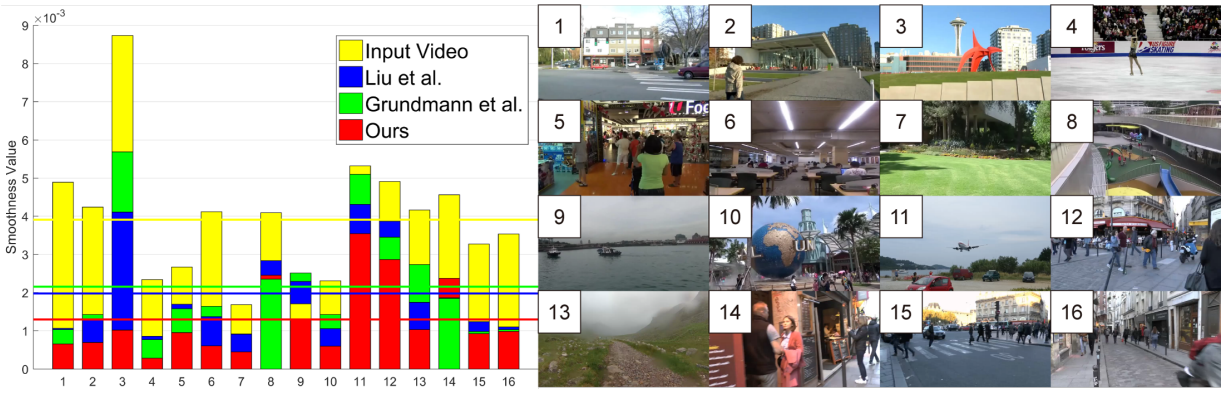


Figure 2.12: *Example video stills from our test set, and smoothness comparison on general videos, showing our result, Liu et al.[LGJA09] result and Grundmann et al.[GKE11] result. Note that a lower bar indicates a better result. Numbers on video stills indicate the example indices on the bar graph. Horizontal lines represent the average values of the corresponding methods.*

problem in general video stabilization. However, Fig. 2.12 shows that our method achieves much better results to Grundmann et al.[GKE11] and Liu et al.[LYTS13] in most of general video cases. Note that we are able to generate more stabilized video than Grundmann et al.[GKE11] and Liu et al.[LYTS13] in examples 1-10, which are taken from their datasets. Moreover, our method performs significantly better in the blurred video case(example 13) since we used optical flow as the pixel tracking method.

Failure Cases Our frame motion model does not apply to videos with complex motions, e.g. strong rolling shutter effect and fisheye effect. We also include a selfie video taken with a fisheye camera in the accompanying video, in which our method does not perform well. Our method does not explicitly correct motion blur. Therefore our results on videos with strong motion blur (mostly because of low illumination) will have unsatisfactory appearance. Our result of example 4 in the selfie video dataset belongs to this category. Note that Fig. 2.9 shows that we still generate better results for example 4 compared to Liu et al.[LGJA09] and Grundmann et al.[GKE11].

2.8 Summary

We proposed a novel video stabilization technique for selfie videos. Our method analyzes the motion of foreground(face and body) using a 3D face model and the motion of background by temporally tracking the pixels using optical flow. We achieve visually and quantatively better results than the state-of-the-art general video stabilization methods. Our method also exhibits robustness under different situations(e.g., large foreground occlusion, blur due to motion or out-of-focus and foreground/background motion mismatch).

Our method requires optical flow to track pixels in the video, and therefore suffers from the overhead of computing optical flow for neighboring frames. Another limitation of our method is that we require that facial landmarks can be detected in most of the frames. In our experiments, we linearly interpolate the head position for frames in which no face was detected. If the faces are undetectable in many consecutive frames, simply interpolating head positions will yield inaccurate estimation of the foreground motion. These limitations can be resolved by applying a more efficient optical flow technique and a more robust facial landmark detector. Our frame motion model does not apply to videos with complex motion. Our method also does not correct motion blur. Therefore for night-time videos or videos taken under dark lighting conditions, our method does not produce satisfactory results.

Since our method utilizes the 3D face model in selfie videos, one future work would be using 3D information to estimate 3D camera motion, so that the 3D video stabilization can be applied to selfie videos with large dynamic occlusions. The 3D face model also enables other future works, including manipulating the shape and expression of the face in selfie videos or high quality 3D reconstruction of face and body from selfie videos.

Our work in this chapter made significant impact in industry. Our idea that using 3D face model to track the foreground motion has inspired subsequent work Steadiface[STWL19] by Google, which has made significant impact to the video stabilization in Google Pixel Phone and

Android. To make their method run on mobile SOC, they made certain trade-off between result quality and computational complexity, e.g. using gyroscope instead of image domain features and using global homography for image warping. In Chapter 3, we will further improve the computational performance of our work in this chapter. Our work in Chapter 3 achieves real-time performance as Steadiface[STWL19], while generating better results than Steadiface [STWL19] and comparable results with the optimization based selfie video stabilization in this chapter.

This chapter is a reformatted version of the material as it appears in "Selfie Video Stabilization," Jiyang Yu and Ravi Ramamoorthi, in IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2019 [YR19b]. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Real-time Selfie Video Stabilization

3.1 Introduction

In Chapter 2, we proposed an optimization based selfie video stabilization pipeline that stabilizes the foreground human face and background. However, the optimization based method requires the entire video sequence to be known at the stabilization time, and the processing speed is often slow compared to playback speed of the video. On the other hand, consumer applications like selfie video stabilization require a significantly fast or even real-time online algorithm to be practical. This rules out most video stabilization algorithms requiring high overhead pre-processing like SFM [LGJA09], optical flow [LYTS14, YR19a, CK20] and future motion information [GKE11, LGW$^+$11]. Another selfie video stabilization work Steadiface [STWL19] was based on our work discussed in Chapter 2. Although their work achieves real-time performance, it only estimates global homography for stabilization and cannot handle non-linear local motions, e.g. rolling shutter. Additionally, their work also requires gyroscope information.

In this chapter, we propose a novel learning based *real-time* selfie video stabilization method. Our method is fully automatic and requires no preprocessing and user assistance. The method is designed to tackle the challenges discussed above. An overview of our method is

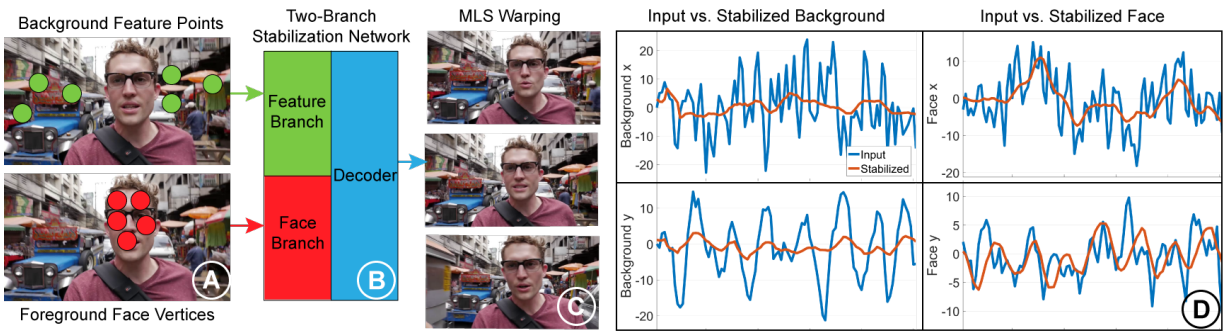


Figure 3.1: *Our method stabilizes selfie videos using Ⓐ background feature points and foreground face vertices in each frame. Ⓑ The two-branch stabilization network infers Ⓒ the moving least squares (MLS) warping for each frame. Ⓓ We show the face and background motion of the input vs. our stabilized result. For visualization only, the background tracks are computed from the translation component of the homography between consecutive frames. The face tracks are computed from the centroid of the fitted face vertices in each frame.*

shown in Fig. 3.1. To achieve real-time performance, our method is purely 2D video stabilization, meaning that our method only depends on the motion of sparse 2D points detected from input video (Fig. 3.1Ⓐ). This makes our method significantly faster than the offline selfie video stabilization in Chapter 2. In the first step, we avoid the occlusion problem by training a segmentation network to infer the foreground regions and remove the feature points in the foreground. To take foreground motion into consideration, we use the 3DDFA [ZLLL19] to fit a 3D mesh to video frames. To warp the original frames into stabilized frames, we use the rigid moving least squares (MLS) [SMW06] (Fig. 3.1Ⓒ). One useful property of MLS warping is that it preserves the original shape of regions that lack warp nodes. In our method, we directly use the background feature points as the warp nodes so that the face shape remains undistorted.

The core of our method is the stabilization network (Fig. 3.1Ⓑ). The network generates the displacement of the warp nodes from the input face vertices and feature points, so that motions of both the foreground (represented by face vertices) and the background (represented by feature points) are minimized. We also design the network structure so that the user can optionally control the degree of stabilization of the foreground and background on the fly. Unlike traditional neural networks that use activation layers to introduce non-linearity, our network only contains

linear convolutional layers to maintain the linearity relation between the input feature point scale and output warp node displacement scale. Although our network ultimately represents a linear relationship between input feature points and the displacement of output warp nodes, direct optimization for this linear relationship is prohibitive in terms of computational efficiency and accuracy (Table 3.3)[1]. Training a linear network instead makes the problem tractable, which is similar to how optimizing over non-linear network weights has regularized optimization problems in video stabilization [YR19a] and other domains [UVL18] in previous works. In Sec. 3.6.4, we will justify this unconventional design by quantitatively comparing the results with the network trained with activation layers.

The contribution includes: 1) A novel selfie video stabilization network that enables real-time selfie video stabilization. Our network directly infers the rigid moving least squares warp from the 2D feature points, stabilizing both the foreground face motion and background feature motion (Sec. 3.3.1 and Sec. 3.3.2). In Sec. 3.4.3 we will show that the structure of our network allows an optional online control of stabilization focus. In Sec. 3.6.4 we will show that our network structure with only linear layers leads to a better result compared to a traditional network structure with activation layers, implying that deep linear neural networks can outperform traditional neural networks in certain scenarios.

2) Grid approximated moving least squares warping that works at a real-time rate. For our method, the MLS algorithm with hundreds of warp nodes requires a significant amount of time to warp a frame. We use a sparse grid to approximate the MLS warping (Sec. 3.5) that improves the warping speed by two orders of magnitude. Our entire pipeline is able to stabilize the video at 26fps.

3) A novel large selfie video dataset with per-frame labeled foreground masks. We will discuss the details of our dataset in Sec. 3.4.1. The dataset enables the training of the foreground detection network and the stabilization network in this chapter. We will make our dataset publicly

[1]Note that the objective function we use is non-linear, so a non-linear optimizer needs to be used in any case, rather than simple linear least squares solvers.

available for face and video related researches.

3.2 Previous Work

While video stabilization has been extensively studied, most of the works belong to the offline video stabilization category. The major reason is that most video stabilization methods rely on temporally global motion information to compute the warping for the current frame. Recent works using global motion information include the L_1 optimal camera paths [GKE11], bundled camera paths [LYTS13], subspace video stabilization [LGW+11], video stabilization using epipolar geometry [GF12], content-preserving warps [LGJA09] and spatially and temporally optimized video stabilization [WLHL13]. These works all involve the detection of feature tracks and smoothing under certain constraints. Some works use optical flow [LYTS14, YR19a] or video coding [LLZZ17] instead of feature tracking as the motion detection method. However, they still inherently require future motion information for the global motion optimization.

One may argue that these global optimization based video stabilization methods can be easily modified to online methods by applying a sliding window scheme. However, note that methods like bundled camera paths [LYTS13] only smooth tracks formed by feature points. Falsely detected features can easily affect the optimization, especially when the window size is small. Moreover, [LYTS13] requires global motion information to achieve the reported result. One can expect performance to decrease if a short sliding window is applied. In Sec. 3.6 we will show that [LYTS13] already generates inferior results than ours using the entire video (Fig. 3.11 and Fig. 3.12). As we will discuss in Sec. 3.4, our pipeline considers all feature points in a window as a whole; the feature points are not only temporally related but also spatially related. Note that this makes the objective function non-linear, thus we cannot simply use the least squares optimization of [LYTS13]. Moreover, our network contains several downsample layers, which effectively blend feature points. This makes our network robust to individual erroneous features,

and it generates satisfactory results with a short 5-frame sliding window.

Deep learning has also been applied to video stabilization in some works. These attempts include using adversarial networks to generate stabilized video directly from unstabilized input frames [XHW+18] and estimate a warp grid from input frames [WYL+19]. These methods are difficult to generalize to videos in the wild. Other learning based works (e.g., [CK20]) iteratively interpolate frames at intermediate positions. These works still require optical flow and are prone to artifacts at moving object boundaries.

Some works are more related to the selfie video stabilization context. Our optimization based selfie video stabilization method in Chapter 2 uses the face centroid to represent the foreground motions while stabilizing the background motions. However, our method in Chapter 2 uses the optical flow to detect the background motion and the foreground mask, which is computationally expensive for real-time applications. The method in Chapter 2 also requires motion information of the entire video, which makes it impractical for online video stabilization. The learning based method introduced in this chapter does not require the dense optical flow computation and does not require future motion information, therefore is more efficient than the method in Chapter 2.

Steadiface [STWL19] is an online real-time selfie video stabilization method. They used facial key points as the reference and the gyroscope information as auxiliary to stabilize human faces. However, their approach uses simple full-frame transformation to warp the frame, which cannot compensate for non-linear distortion like rolling shutter. Our method uses grid-based MLS warping which provide flexibility to handle non-linear distortions. Our method also models the face motion more accurately using a face mesh instead of face landmarks in [STWL19]. Due to these limitations, Steadiface [STWL19] will not produce results comparable with ours by simply adding a hyperparameter to control foreground and background stabilization like our method. We will show that the quality of our results is significantly better than Steadiface [STWL19] in Fig 3.13(b) and the supplementary video.

Table 3.1: Notations in this Chapter

Symbols	Explanation
t	Frame index
$\mathbf{M}_t$	Foreground mask
$\mathbf{P}_t$	Background feature points
$\mathbf{Q}_t$	Correspondence of $\mathbf{P}_{t-1}$ in frame t
$\mathbf{F}_t$	Face vertices
$\widehat{\mathbf{Q}}_t$	Target coordinate of $\mathbf{Q}_t$
$\mathbf{v}$	Coordinate of a pixel
$W(\mathbf{v};\mathbf{Q}_t,\widehat{\mathbf{Q}}_t)$	Rigid MLS warping function
$\widehat{\mathbf{v}}$	Warped coordinate of pixel $\mathbf{v}$
$\mathbf{q}_i$	ith column of $\mathbf{Q}_t$
w_i	MLS weight of $\mathbf{q}_{i,t}$ to pixel $\mathbf{v}$
α	MLS parameter
$\mathbf{c}$	Weighted centroid of $\mathbf{Q}_t$
$\widehat{\mathbf{c}}$	Weighted centroid of $\widehat{\mathbf{Q}}_t$
$\mathbf{q}_i^*$	Vector from $\mathbf{c}$ to $\mathbf{q}_{i,t}$
$\widehat{\mathbf{q}}_i^*$	Vector from $\widehat{\mathbf{c}}$ to $\widehat{\mathbf{q}}_i$
$\mathbf{A}_i$	Transformation matrix of $\widehat{\mathbf{q}}_i^*$
$\mathbf{g}_j$	jth grid vertex
$\mathbf{G}$	Grid vertices enclosing $\mathbf{v}$
$\mathbf{D}$	Bilinear weights of $\mathbf{v}$ with respect to $\mathbf{G}$

MeshFlow [LTY+16] is an online real-time general video stabilization method. They use a sparse grid and feature points to estimate the dense optical flow. However, as a general video stabilization method, they do not consider the foreground/background motion and the large occlusion imposed by the face and body. This reduces the robustness in the context of selfie videos.

In Sec. 3.6, we will compare our result with selfie video stabilization [YR19b](our method in Chapter 2), Steadiface [STWL19], MeshFlow [LTY+16] and the state-of-the-art learning based approaches [WYL+19, CK20]. We also compare with the bundled camera path video stabilization [LYTS13] representing a typical offline general video stabilization method as the reference.

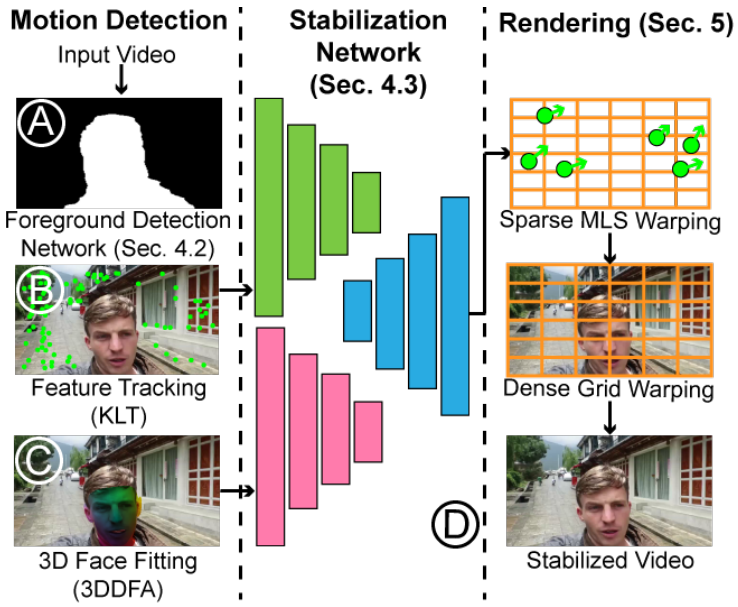


Figure 3.2: *The pipeline of our method. Ⓐ We first detect the foreground regions of the input video frame. Ⓑ The motion of the background is tracked using feature points. Ⓒ The foreground motion is tracked using 3D face vertices. Ⓓ We train a stabilization network to infer the displacement of the MLS warp nodes. Finally, we use a grid to approximate the MLS warping and generate the stabilized frame.*

3.3 Overview of algorithm pipeline

Our pipeline is shown in Fig. 3.2. The pipeline consists of three parts: motion detection, stabilization and warping. In this section, we will introduce these parts separately and provide an overview of the selfie video stabilization process. We summarize the notations used in this chapter in Table. 3.1. The training of the neural networks mentioned below will be discussed in Sec. 3.4.

3.3.1 Motion Detection

As discussed in Sec. 3.1, for selfie videos, we seek to stabilize the foreground and background at the same time. Therefore, both the motion of the face and the background need to be detected. To distinguish the foreground and the background, we first use a pre-trained foreground detection network to infer a foreground mask $\mathbf{M}_t$ where $\mathbf{M}_t = 1$ represents the

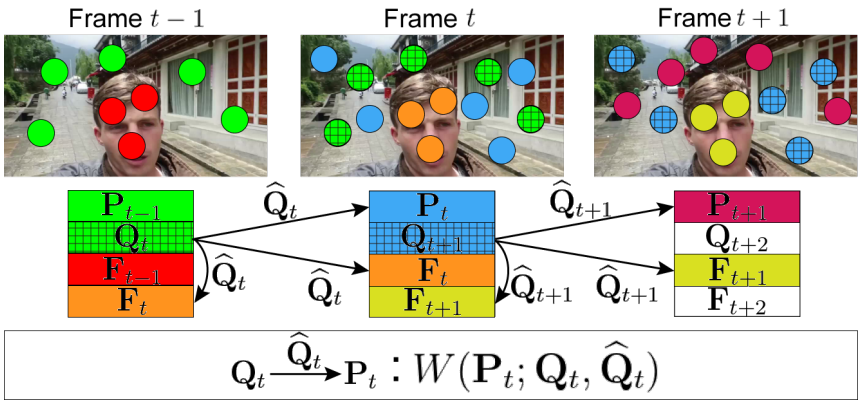


Figure 3.3: *The warping strategy of our method. In the shown frames, the background feature points marked with the same color are in correspondence. The feature points marked with grid patterns are the warp nodes. The arrows represent the MLS warping operation. During the stabilization, both the feature points* $\mathbf{P}_t$ *and the face vertices* $\mathbf{F}_t$ *are warped by the warp nodes* $\mathbf{Q}_t$.

foreground region of frame t. We show a sample foreground mask in Fig. 3.2Ⓐ. The details regarding the foreground detection network will be discussed in Sec. 3.4.2. For the background region where $\mathbf{M}_t = 0$, we use the Shi-Tomasi corner detector[ST94] to detect feature points in a frame and the KLT tracker to find their correspondences in the next frame, as shown in Fig. 3.2Ⓑ. We uniformly sample 512 feature points for each frame, since fewer feature points cannot provide enough coverage of frame regions and more feature points will make the pipeline less efficient without significant improvement in warping quality. We will visually compare the different number of feature point selections in Sec. 3.6. We denote the selected feature points in frame t as $\mathbf{P}_t \in \mathbb{R}^{2\times 512}$. Their correspondences in frame $t+1$ are denoted as $\mathbf{Q}_{t+1} \in \mathbb{R}^{2\times 512}$.

To detect the motion of the foreground, we fit a 3D face mesh to each frame using 3DDFA proposed in [ZLLL19]. An example of a fitted 3D face mesh is shown in Fig. 3.2Ⓒ. As in the background, we uniformly sample 512 face vertices to represent the face position in a frame. Furthermore, we only consider the 2D projection of the face mesh in our method. In this chapter, we denote the selected face vertices as $\mathbf{F}_t \in \mathbb{R}^{2\times 512}$, where t represents the frame index.

ALGORITHM 1: The rigid MLS warping algorithm $W(\mathbf{v};\mathbf{Q},\widehat{\mathbf{Q}})$

Input : Source coordinates of a pixel $\mathbf{v}$, source node coordinates $\mathbf{Q}$ and target node coordinates $\widehat{\mathbf{Q}}$

Output : Target coordinates of a pixel $\widehat{\mathbf{v}}$

for $i \leftarrow 1$ **to** *512* **do**

  $w_i = 1/|\mathbf{v}-\mathbf{q}_i|^{2\alpha}$

end

$\mathbf{c} = \left(\sum_{i=1}^{512} w_i\mathbf{q}_i\right) / \left(\sum_{i=1}^{512} w_i\right) \quad \widehat{\mathbf{c}} = \left(\sum_{i=1}^{512} w_i\widehat{\mathbf{q}}_i\right) / \left(\sum_{i=1}^{512} w_i\right)$

for $i \leftarrow 1$ **to** *512* **do**

  $\mathbf{q}_i^* = (\mathbf{q}_i-\mathbf{c})^T \quad \widehat{\mathbf{q}}_i^* = (\widehat{\mathbf{q}}_i-\widehat{\mathbf{c}})^T$

  $\mathbf{A}_i = w_i\begin{pmatrix}\mathbf{q}_i^* \\ -\mathbf{q}_i^{*\perp}\end{pmatrix}\begin{pmatrix}\mathbf{v}-\mathbf{c} & -(\mathbf{v}-\mathbf{c})^\perp\end{pmatrix}$,

  where $\perp$ is an operator on 2D vector $(x,y)^\perp = (-y,x)$

end

$\widehat{\mathbf{v}} = |\mathbf{v}-\mathbf{c}|\left(\sum_{i=1}^{512}\mathbf{A}_i\widehat{\mathbf{q}}_i^*\right) / \left|\sum_{i=1}^{512}\mathbf{A}_i\widehat{\mathbf{q}}_i^*\right| + \widehat{\mathbf{c}}$

3.3.2 Stabilization

To stabilize the video, we use the rigid moving least square(MLS) warping[SMW06] to warp the frames. In Fig. 3.3, we depict the warping strategy of a video sequence. The moving least square warping requires a set of warp nodes for each frame t. We use the correspondences of detected feature points, i.e., $\mathbf{Q}_t$, as the warp nodes for frame t (marked by green grid dots in Fig. 3.3). Besides all the pixels in frame t, the feature points $\mathbf{P}_t$ (blue dots) and the face vertices $\mathbf{F}_t$ (orange dots) are also warped by $\mathbf{Q}_t$ during the stabilization to reflect the change of their positions.

Denote the target location of the warp nodes as $\widehat{\mathbf{Q}}_t$, then the rigid MLS warping operation (shown as the arrows in Fig. 3.3) can be written as a function $W(\mathbf{v};\mathbf{Q}_t,\widehat{\mathbf{Q}}_t)$, where $\mathbf{v}$ is a pixel/feature point/face vertex to be warped. Denoting each column of a matrix $\mathbf{Q}_t$ as $\mathbf{q}_{i,t} \in \mathbb{R}^{2\times 1}$ where $i \in [1,512]$, the rigid MLS warping procedure is defined by a series of computations in Algorithm 1. Since the MLS warping is not related to the time dimension, we omit the time subscript t for simplicity. In Algorithm 1, we use relatively small $\alpha = 0.3$ to maintain a smooth warp field and avoid artifacts.

In this chapter, we propose a convolutional neural network (Fig. 3.2(D)) to infer the displacements of warp nodes $\widehat{\mathbf{Q}}_t - \mathbf{Q}_t$. In Sec. 3.4.3, we will discuss the training of this stabilization network.

3.3.3 Warping

Although the MLS warping can achieve real-time warping with a relatively small number of warp nodes, in our application, warping with hundreds of warp nodes is both time and memory inefficient. With our implementation of GPU accelerated MLS warping, with 512 warp nodes, a frame of size 448×832 must be divided into 16 blocks in order to be fit in a NVIDIA 2080Ti GPU's memory and the warp speed is approximately 1s/frame. This makes it prohibitive for real-time applications. To address this issue, we use a grid to approximate the MLS warp field. This approximation enables real-time performance of our method and yields high-quality visual results. In Sec. 3.5, we will demonstrate the details of the grid warping approximation.

3.4 Network

In this section, we discuss the details regarding the stabilization network and foreground detection network. We first present our novel selfie video dataset (Sec. 3.4.1), then discuss details of the foreground detection network (Sec. 3.4.2) and stabilization network (Sec. 3.4.3). Finally, we introduce a sliding window scheme to apply our stabilization network to arbitrarily long videos (Sec. 3.4.4).

3.4.1 Dataset

Although large scale video datasets like Youtube-8M [AEHKL$^+$16] have been widely used, public videos with continuous presence of faces are difficult to collect. We propose a novel selfie video dataset containing 1005 selfie video clips, which is significantly larger than

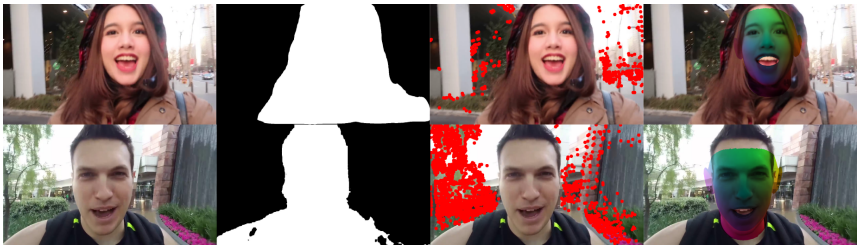


Figure 3.4: *Our selfie video dataset. From left to right: color frame, ground truth foreground mask, background feature points, face mesh.*

existing selfie video datasets proposed in [YR19b](33 videos) and [LCSP19](80 videos). We first manually collect long vlog videos captured with mobile devices from the Internet. In these videos, we aim to locate the clips that have stable face presence. We use the face detector from Dlib [Kin09] to detect faces in each frame, and maintain a global counter to count the number of consecutive frames that contain faces. If the face can be detected in more than 50 consecutive frames, we cut the raw video into a new clip. In addition to the regular color videos, our dataset also includes a ground truth foreground mask for each frame. We manually label the foreground region of the first frame of each video clip, then use Siammask_E [CT19] to track the foreground object and generate the foreground mask for the video clip. In addition, we also provide the detected feature points in each frame and their correspondences in the next frame. Finally, for each frame, we provide the dense 3D face mesh fitted using [ZLLL19]. In Fig. 3.4, we show some video stills, the corresponding foreground masks, the background feature points and the 3D face mesh from our dataset. Our dataset will be made publicly available upon publication.

3.4.2 Foreground Detection Network

Since we have the ground truth mask for our selfie video dataset, training a binary segmentation network is straightforward. We train an FCN8s network proposed in [LSD15] for this segmentation task. Although there are more advanced structure for segmentation [NSR18, CKR+19], we find that FCN8s achieves satisfactory results for our application. The input of the network is the raw RGB frame, and the output is the binary segmentation mask M mentioned

Figure 3.5: *Examples of the foreground mask detected with our trained foreground detection network.*

in Sec. 3.3.2. The training uses Adam optimizer with a 10^{-3} learning rate and a binary cross entropy loss. Figure 3.5 provides examples of the inferred masks on video frames outside our dataset. Note that the inferred mask does not perfectly indicate the foreground region, but it is accurate enough to distinguish the foreground and the background.

3.4.3 Stabilization Network

For a video with T frames, we are able to detect $T-1$ groups of feature points $\mathbf{P}_t$ and their correspondences in the next frame $\mathbf{Q}_{t+1}$ using the KLT tracking mentioned in Sec. 3.3.1. For each frame, we seek to infer the displacement of warp nodes $\widehat{\mathbf{Q}}_t - \mathbf{Q}_t$ so that the overall motion of the video is minimized. Formally, the loss function for the background can be written as

$$L_b = \sum_{t=1}^{T-1} \left\| W(\mathbf{P}_t; \mathbf{Q}_t, \widehat{\mathbf{Q}}_t) - \widehat{\mathbf{Q}}_{t+1} \right\|_2 \tag{3.1}$$

where $W(\mathbf{P}_t; \mathbf{Q}_t, \widehat{\mathbf{Q}}_t)$ is the MLS warping function as mentioned in Sec. 3.3.2. Note that here we apply the MLS warping function to a group of feature points, i.e., each column of $\mathbf{P}_t$ are treated as the coordinates of a pixel and warped by all the warp nodes according to Algorithm 1. Since the $\mathbf{P}_t$'s correspondence $\mathbf{Q}_{t+1}$ are the warp nodes for the next frame, so here we should directly use their new position $\widehat{\mathbf{Q}}_{t+1}$.

Similarly, we can also define the foreground loss function using the face vertices:

$$L_f = \sum_{t=1}^{T-1} \left\| W(\mathbf{F}_t; \mathbf{Q}_t, \widehat{\mathbf{Q}}_t) - W(\mathbf{F}_{t+1}; \mathbf{Q}_{t+1}, \widehat{\mathbf{Q}}_{t+1}) \right\|_2 \tag{3.2}$$

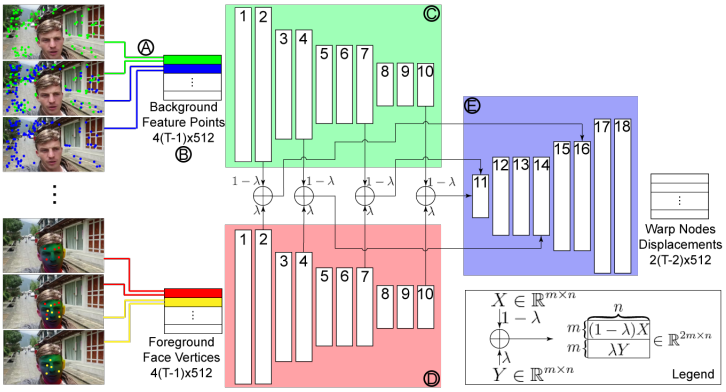
In this equation, the difference with Eq. 3.1 is that the face vertices in the next frame $t+1$ are warped by the warp nodes $\mathbf{Q}_{t+1}$.

We also introduce a value λ to control the weighting of foreground stabilization and background stabilization. The complete loss function is defined as:

$$L = (1-\lambda)L_b + \lambda L_f \tag{3.3}$$

In Eq. (3.3), the value $\lambda \in (0,1)$ controls the stabilization focus on foreground versus background. A larger λ means that we tend to stabilize the face more, and a smaller λ means we tend to stabilize the background more. Our method uses $\lambda = 0.3$ by default and stabilizes the video automatically. The user can also change the value online during the stabilization. In the supplementary video, we will show an example of our network seamlessly handling the changing λ during the stabilization.

Network Structure Our network structure is inspired by the 2D autoencoder network structure. However, there are two major problems to solve before the 2D autoencoder can be used in the context of selfie videos. First, the input dimension does not match the network structure: we only have sparse feature points instead of a dense optical flow image. Second, the vanilla autoencoder does not provide control over the foreground and background stabilization. To solve these problems, we design our network as a 1D autoencoder with two input branches. We demonstrate our network structure in Fig. 3.6. For simplicity, we will omit the batch dimension in the discussion. For each frame, the feature points $\mathbf{P}_t \in \mathbb{R}^{2\times 512}$ and $\mathbf{Q}_t \in \mathbb{R}^{2\times 512}$ mentioned in Sec. 3.3.1 are concatenated in the row dimension, resulting in a frame feature tensor $\mathbf{X}_t \in \mathbb{R}^{4\times 512}$ as shown in Fig. 3.6Ⓐ. We concatenate the frame feature tensor of $T-1$ frames, forming the feature branch input tensor $\overline{\mathbf{X}} \in \mathbb{R}^{4(T-1)\times 512}$ shown in Fig. 3.6Ⓑ. Similarly, we concatenate the



Index	Layer Type	Input Size	Output Size	Kernel Size	Stride	Dilation	Padding
1	Conv1d	4(T-1)x512	Cx512	3	1	1	1
2	Conv1d	Cx512	2Cx256	4	2	1	1
3	Conv1d	2Cx256	2Cx256	3	1	1	1
4	Conv1d	2Cx256	4Cx128	4	2	1	1
5	Conv1d	4Cx128	4Cx128	3	1	1	1
6	Conv1d	4Cx128	4Cx128	3	1	1	1
7	Conv1d	4Cx128	8Cx64	4	2	1	1
8	Conv1d	8Cx64	8Cx64	3	1	2	2
9	Conv1d	8Cx64	8Cx64	3	1	2	2
10	Conv1d	8Cx64	8Cx64	3	1	2	2
11	ConvTranspose1d	32Cx64	8Cx128	4	2	1	1
12	Conv1d	8Cx128	8Cx128	3	1	1	1
13	Conv1d	8Cx128	8Cx128	3	1	1	1
14	ConvTranspose1d	16Cx128	4Cx256	4	2	1	1
15	Conv1d	4Cx256	4Cx256	3	1	1	1
16	ConvTranspose1d	8Cx256	2Cx512	4	2	1	1
17	Conv1d	2Cx512	2Cx512	3	1	1	1
18	Conv1d	2Cx512	2(T-2)x512	1	1	1	0

Figure 3.6: *Our stabilization network structure. On the left we show a sequence of input frames.* Ⓐ *The feature points and their correspondences in the next frame are concatenated as a* 4×512 *tensor.* Ⓑ *The tensors in the same window are concatenated to a large* $4(T-1) \times 512$ *tensor. The same operation is done for face vertices. The output of* Ⓒ *the feature branch and* Ⓓ *the face branch of our network are weighted by* λ *and concatenated.* Ⓔ *The decoder outputs the displacements of the warp nodes. The layer parameters are listed on the right hand side.*

face vertices into the face branch input tensor $\overline{\mathbf{Y}} \in \mathbb{R}^{4(T-1)\times 512}$. Tensor $\overline{\mathbf{X}}$ and $\overline{\mathbf{Y}}$ are encoded separately with 1D convolutional layers (Figs. 3.6Ⓒ and Ⓓ), which only convolve with the last dimension of the tensors. The number of filters in each layer is multiple of a base number C. The resulting output tensor sizes are noted in the table on the right in Fig. 3.6. The encoded tensor from different downsample levels are weighted by λ and concatenated for skip connection to decoders (Fig 3.6Ⓔ), so that the stabilization of foreground and background can be controlled by the user input λ. Note that the order of feature points does not affect the network, since we train the network with randomly sampled feature points and face vertices and the encoder downsamples the input and essentially blends the feature points regardless their original order. The decoder generates the displacements of the warp nodes. Note that for a length T video, we do not warp the first frame and last frame. The reason is that the goal of video stabilization is to smooth the original motion, not to eliminate the motion. Our network is effectively inferring the warp field for the intermediate $T-2$ frames and stabilizes the video instead of aligning all the frames.

Linear Network Our network does not contain activation layers, which is different from conventional neural networks. Conventional neural networks contain activation layers to introduce

non-linearity. However, in Eq. (3.1) and Eq. (3.2), we define the loss function directly on feature points detected in the image. This physically based definition requires the linear relationship between the input and the output of the stabilization network. Intuitively, N times larger feature point coordinates indicates N times longer motion vector, therefore the output displacement that compensates the motion should also be N times larger. We conducted experiments on the network with activation layers added. In Table 3.2, we will show that our linear network produces better results than the network with activation layers.

Note that the linear relationship between input and output can be posed as a matrix-vector product, i.e., $\mathbf{n} = \mathbf{A}\mathbf{m}$ where $\mathbf{A} \in \mathbb{R}^{1024(T-1)\times 4096(T-1)}$ is a large matrix that transforms concatenated and reshaped input feature points and face vertices $\mathbf{m} \in \mathbb{R}^{4096(T-1)\times 1}$ to reshaped warp node displacements $\mathbf{n} \in \mathbb{R}^{1024(T-1)\times 1}$. The optimization problem equivalent to our network training can be defined as:

$$\min_{\mathbf{A}} L(\mathbf{m}, \mathbf{n}), \tag{3.4}$$

where L is the loss function defined in Eq. 3.3. Solving this problem directly is difficult and prohibitive in the video stabilization for the following reasons. First, the matrix $\mathbf{A}$ is dense and the problem is highly under-determined. Second, the loss function we defined involves non-linear moving least squares warping; the problem cannot be solved using a simple linear system solver as in [LYTS13]. Finally, the problem has to be solved for each sliding window in the online video stabilization, making it impossible to achieve real-time performance. On the other hand, the linear neural network has two advantages compared to posing the problem as an optimization. First, the convolutional layers contain only small kernels; the concatenation of layers is equivalent to decomposing the dense matrix into a series of sparse matrices which is easier to solve through backpropagation and gradient descent. Second, the network implicitly provides regularization by training on a large dataset; using a pretrained network avoids the overfitting problem in the optimization and also enables computational real-time performance.

Another way to pose the stabilization process as an optimization problem is to directly

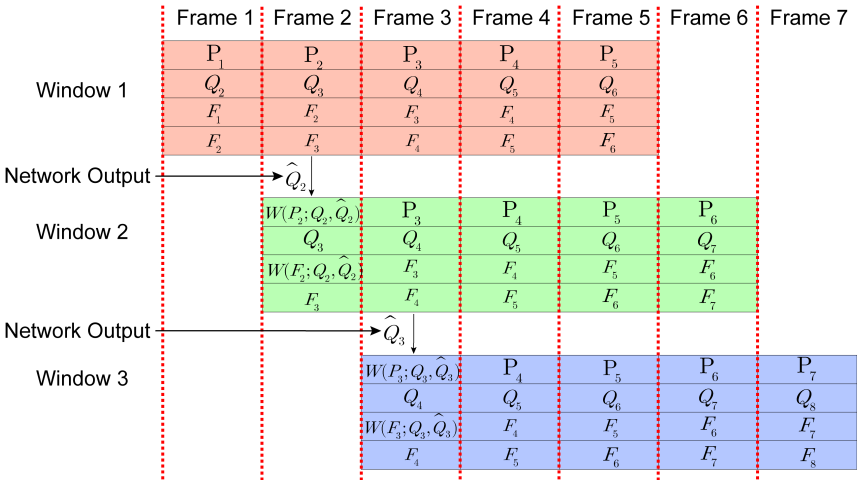


Figure 3.7: *The sliding window scheme of our method. The inputs of our network for each window are marked with the same color. For each window, the second frame is stabilized. The background feature points and the foreground face vertices are updated accordingly and become the next window's input.*

solve for the warp node displacement $\widehat{\mathbf{Q}}_t - \mathbf{Q}_t$ to minimize the non-linear loss function L. Although this formulation is tractable, it suffers from the overfitting problem since our feature points are sparse. Using this formulation in real-time video stabilization is also prohibitive due to its speed, since we need to conduct non-linear optimization for each sliding window. We will discuss its results in detail in Table 3.3.

Training To train the stabilization network, we randomly draw a length T segment of selfie video from our dataset. The feature points and face vertices in each frame are perturbed by random affine transformation with rotation between $[-10°, 10°]$ and translation between $[-50, 50]$ except the first frame and the last frame. We also generate a random λ value between $(0, 1)$. The training uses Adam optimizer with a 10^{-4} learning rate. The loss function is defined in (3.3).

3.4.4 Sliding Window

Since the stabilization network only takes fixed length video segments, to apply to arbitrary length selfie videos, we apply a sliding window scheme. We demonstrate our sliding window scheme in Fig. 3.7. Each window is marked by the same color, which is the input to our network

for the window. Consider window 1 as an example. The outputs of our stabilization network are the displacements of the warp nodes $\widehat{\mathbf{Q}}_2$, $\widehat{\mathbf{Q}}_3$ and $\widehat{\mathbf{Q}}_4$ as we discussed in the network structure. To stabilize frame 2, we use the MLS warp function $W(\mathbf{v};\mathbf{Q}_2,\widehat{\mathbf{Q}}_2)$ to warp frame 2. We then warp the feature points and face vertices using $W(\mathbf{P}_1;\mathbf{Q}_1,\widehat{\mathbf{Q}}_1)$ and $W(\mathbf{F}_1;\mathbf{Q}_1,\widehat{\mathbf{Q}}_1)$, since warping the frame leads to updated positions of the original feature points and face vertices. The updated feature points and face vertices become a part of window 2, which is the next window starting at frame 2. In our experiment, we use a sliding window with length $T = 5$.

3.5 Warping Acceleration

As discussed in Sec. 3.3, using the MLS warping with 512 warp nodes in our case is impractical for real-time application. To accelerate the warping speed, for the final rendering of the frame, we use a grid to approximate the warp field generated by MLS warping. Denote a grid vertex in frame t by $\mathbf{g}_j \in \mathbb{R}^{2\times 1}$, where j is the index of grid vertices. Each pixel $\mathbf{v}$ can be defined by the bilinear interpolation of the enclosing four grid vertices, denoted by $\mathbf{G} \in \mathbb{R}^{2\times 4}$:

$$\mathbf{v} = \mathbf{GD} \tag{3.5}$$

where $\mathbf{D} \in \mathbb{R}^{4\times 1}$ is the vector of bilinear weights.

In the first step of rendering, we warp the grid vertices with warp nodes $\mathbf{Q}_t$ and their target coordinates $\widehat{\mathbf{Q}}_t$:

$$\widehat{\mathbf{g}}_j = W(\mathbf{g}_j;\mathbf{Q}_t,\widehat{\mathbf{Q}}_t) \tag{3.6}$$

Since the grid vertices are sparse, warping with MLS is computationally efficient. We then densely warp the pixels $\mathbf{v}$ using the MLS warped grid coordinates:

$$\widehat{\mathbf{v}} = \widehat{\mathbf{G}}\mathbf{D} \tag{3.7}$$

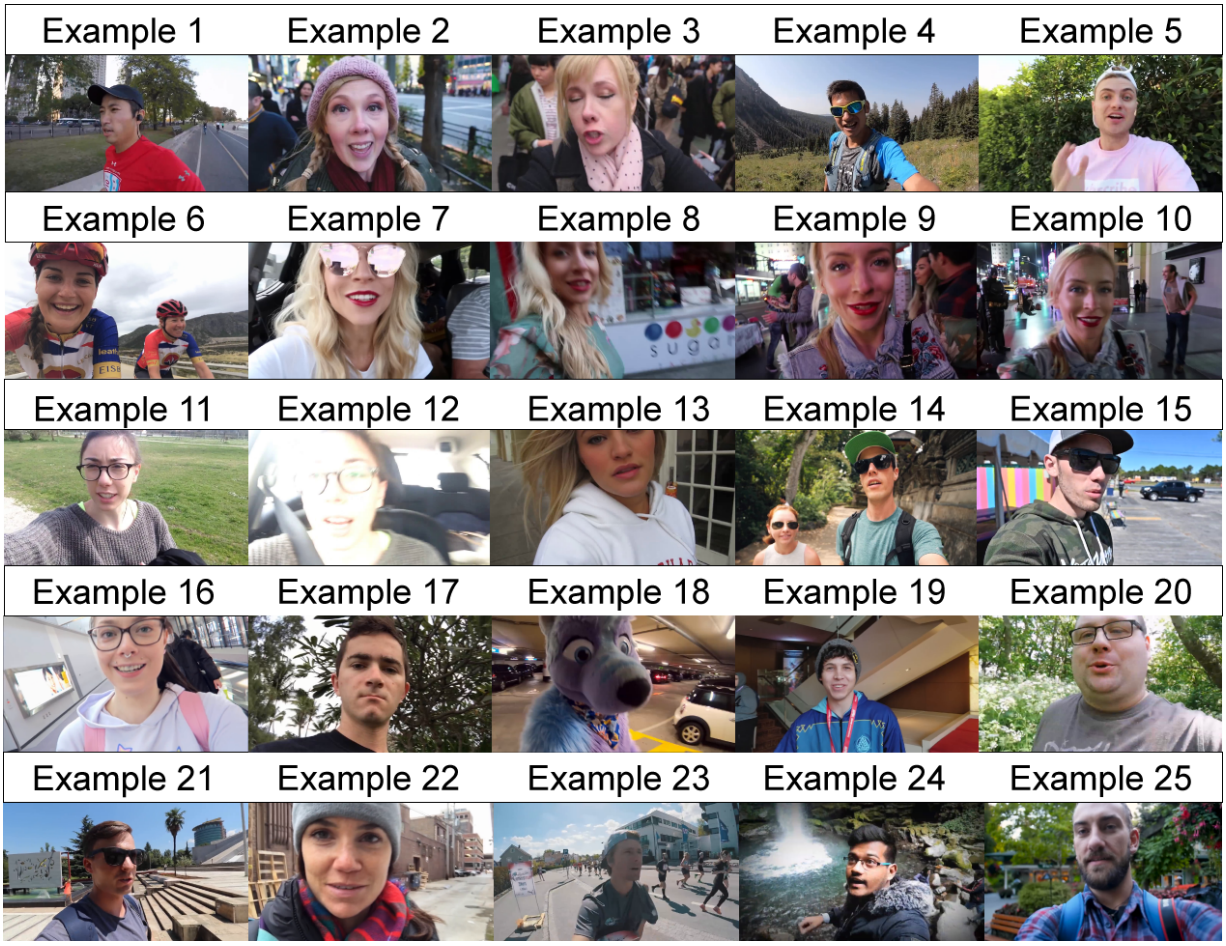


Figure 3.8: *The 25 selfie video examples used for testing, referred to in Sec. 3.6. These videos are selected to cover a variety of challenging scenarios in real applications.*

where $\widehat{\mathbf{G}}$ consists of the transformed enclosing four grid vertices $\widehat{\mathbf{g}}_j$. This step contains only one matrix operation, which can be computed at a real-time rate. In our experiment, we find the difference between the results generated with the dense MLS warping and our grid approximation is negligible. Our method is also not sensitive to the selection of the grid size. In our experiment, we use a grid with size 20×20. We implemented the grid warping on GPU by parallel sampling the grid with a pixel-wise dense grid, generating a dense warp field. We then use the dense warp field to sample the video frame, generating the warped frame. Our implementation of this process takes approximately 4ms/frame, compared to the 1s/frame ground truth dense MLS warping.

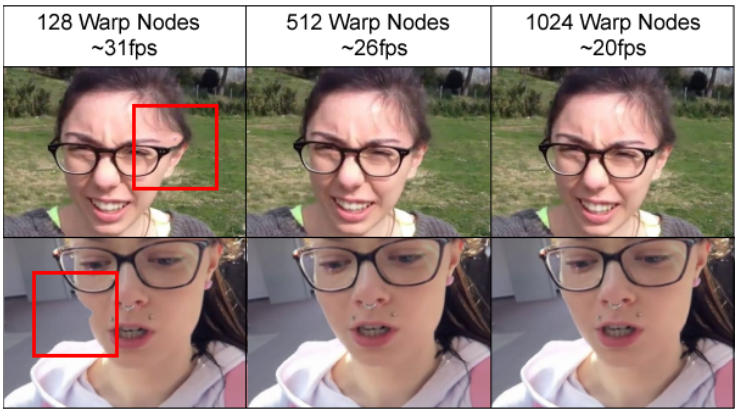


Figure 3.9: *The visual comparison and stabilization speed comparison of different number of warp nodes in our method. The artifacts are marked by the red box.*

3.6 Results

In this section, we present the results of our method. Note that our dataset is cut from a small number of long vlog videos, therefore the faces are from a limited number of people. Some videos in our dataset also do not actually need to be stabilized (e.g., still camera video). To show the effectiveness and the ability of generalization of our method, we collect 25 new selfie videos for testing that contain a variety of challenging scenarios in real applications, and are completely separate from our training dataset. These testing examples are shown in Fig. 3.8. The background scenes vary from indoor (example 16, 18, 19), inside of cars (example 7, 12), city (example 1, 2, 8, 9, 10, 13, 15, 21, 22, 23), crowd (example 2, 3, 9, 10, 16, 23, 24) and wild (example 4, 5, 6, 11, 14, 17, 20, 24, 25). Some of these videos are selected since their content is technically challenging. These challenges include lack of background features (example 6, 7, 12, 15), dynamic background (example 2, 3, 9, 10, 16, 23, 24), sunglasses (example 4, 7, 14, 15, 21), large foreground occlusion (example 13, 16, 20, 22), face cannot be detected or incomplete face (example 8, 9, 13, 16, 18, 20, 22), multiple faces (example 6, 14) and intense motions (example 1, 23). Since the dynamics cannot be shown through video stills, we recommend readers to watch our supplementary video. In the supplementary video, we show the example video clips and our stabilized result side by side. As mentioned in Sec. 3.2, we also provide visual

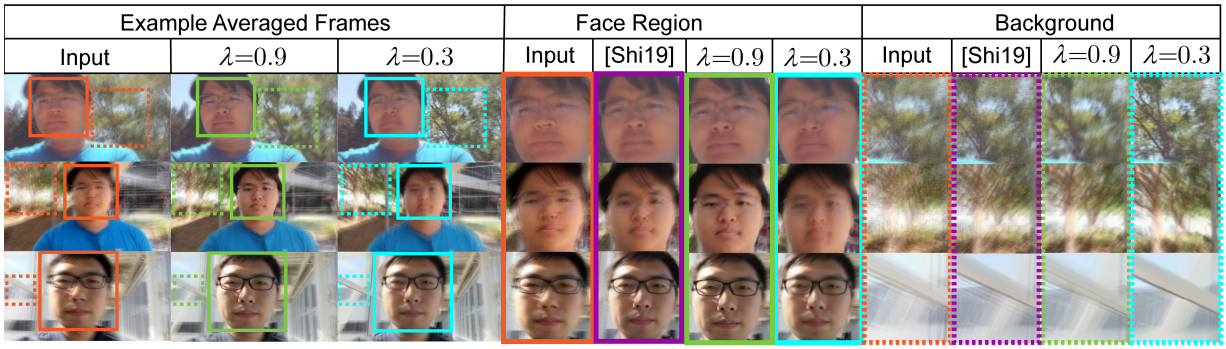


Figure 3.10: *The visual comparison of different values of λ in our method and the state-of-the-art real-time face stabilizaiton method Steadiface [STWL19] using the example videos provided in their work. The images shown are the average of 15 consecutive frames. The face regions and the background regions of the input, the corresponding regions of Steadiface [STWL19] and our method are shown in the insets on the right.*

and quantitative comparison with the offline selfie video stabilization [YR19b](Chapter 2), the real-time selfie video stabilization Steadiface [STWL19], the real-time general video stabilizaion MeshFlow [LTY+16], the offline general video stabilization bundled camera paths [LYTS13] and the state-of-the-art learning-based methods [CK20] and [WYL+19]. Since our examples do not contain gyroscope data, we compare with Steadiface [STWL19] using only the examples provided in their paper.

3.6.1 Pipeline Parameters

The number of warp nodes (feature points) The computational performance of our method greatly depends on the number of warp nodes. Note that we use the feature points as the warp nodes, therefore the number of warp nodes is equivalent to the number of feature points. In the motion detection stage of our pipeline, tracking more feature points requires more processing time, leading to slower stabilization speed. However, if the warp nodes are too sparse in the frame, the possibility of local distortion increases. We provide the average per-frame stabilization time using 128, 512 and 1024 warp nodes and the corresponding warped frames in Fig. 3.9. In Fig. 3.9, using 128 warp nodes results in distortion near the foreground/background bundaries. This is

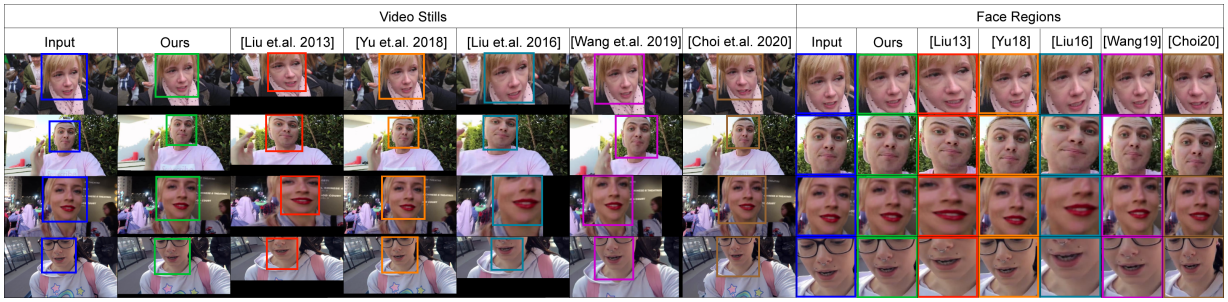


Figure 3.11: *The visual comparison of bundled camera paths [LYTS13], selfie video stabilization [YR19b](Chapter 2), MeshFlow [LTY+16], deep online video stabilization [WYL+19], deep iterative frame interpolation [CK20] and our method. The details of the face regions are shown in the insets on the right. We recommend readers to zoom in and observe the details in the images.*

because in the MLS warping, the warp nodes are implicitly constrained by each other. Fewer constraints reduce the robustness of the warping. An isolated warp node, if tracked mistakenly, introduces local distortion. In our experiment, we select 512 warp nodes since it is a good balance between computational speed and warp quality.

Value of λ In Fig. 3.10 we show the effect of different values of λ. We stabilize the same video clip with λ set to 0.3 and 0.9 respectively. To show the steadiness of the result, we average 15 consecutive frames of the stabilized video. The less blurry the averaged region is, the more stable the region in the stabilized result. For $\lambda = 0.9$, the face regions are less blurry as shown in the green inset, indicating that our network automatically focuses on stabilizing the face. If we set $\lambda = 0.3$, the background regions are less blurry as shown in the cyan inset meaning that the background is more stable. In our experiment, we use a default value of $\lambda = 0.3$, meaning that we stabilize both foreground and background while mainly focusing on the background.

3.6.2 Visual Comparison

We show sample frames from our examples and the stabilized results in Fig. 3.11. Our method stabilizes the frames without introducing visual distortions. The real-time general video stabilization method [LTY+16] and offline general video stabilization method [LYTS13] usually

produce artifacts on the face, since they do not distinguish the foreground and the background. Selfie videos are also challenging for the optical flow estimation in MeshFlow [LTY+16], since the motion within a mesh cell can be significantly different due to the foreground occlusion. The learning based method [WYL+19] generally does not produce local distortions, but tends to generate unstable output video. Due to the accuracy issue in optical flow and frame interpolation, the other learning based method [CK20] generates artifacts, especially near the occlusion boundaries like face boundaries. These artifacts are more obvious when observed dynamically in videos. We recommend the readers to watch the supplementary video for better visual comparison. We also achieve the same quality visual results as the previous optimization based selfie video stabilization proposed in Chapter 2. However, our method is learning-based and runs at the real-time speed, which is orders of magnitude faster compared to their method as we will discuss in Sec. 3.6.7.

We also test our method on the examples in Steadiface [STWL19], which is the state-of-the-art real-time face stabilization method. The images shown on the left of Fig. 3.10 are the average consecutive 15 frames of their results. If we set $\lambda = 0.9$ in our method (mainly stabilize the face), we are able to achieve better face alignment without using the gyroscope information. In addition, we can alternatively set $\lambda = 0.3$ in the stabilization network. The background becomes significantly more stable than the Steadiface [STWL19] results and our $\lambda = 0.9$ results in the averaged frames, indicating that our method is capable of stabilizing the background. Figure 3.10 also indicates that stabilizing the background ($\lambda = 0.3$) leads to a slight sacrifice of face stability, since the motion of the foreground and background is different. In our supplementary video, we will show that this loss of face stability is visually unnoticeable.

3.6.3 Quantitative comparison

We use the three quantitative metrics proposed in [LYTS13] to evaluate the frame size preservation (Cropping), visual distortion (Distortion) and steadiness (Stability) of the stabilization result. Note that since Steadiface [STWL19] require gyroscope information to stabilize the video,

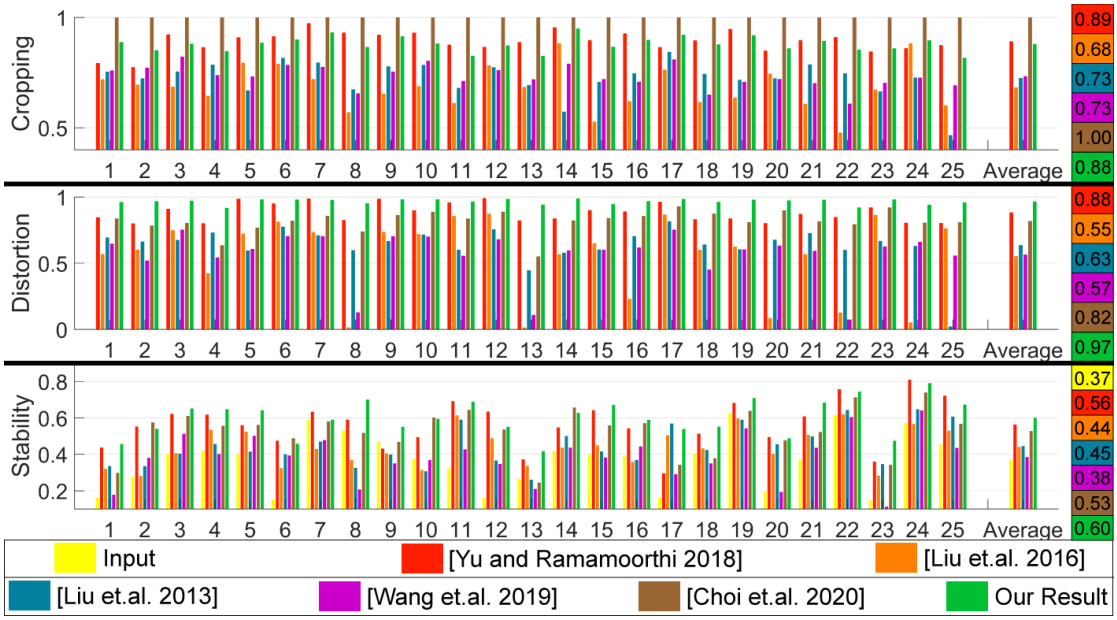


Figure 3.12: *Quantitative comparison of bundled camera paths [LYTS13], selfie video stabilization [YR19b](Chapter 2), MeshFlow [LTY+16], deep online video stabilization [WYL+19], deep iterative frame interpolation [CK20] and our method. In these metrics, a larger value indicates a better result. The x-axis represents the indexes of the example videos listed in Fig. 3.8. The average values over all the example videos are listed on the right.*

the quantitative comparison with their method is conducted using their videos and will be discussed in Fig. 3.13Ⓑ.

In the top row of Fig. 3.12, we show the cropping metric comparison. A larger value represents a larger frame size of the stabilized result. Although [YR19b] uses second order derivative objective, their final frame size is limited by the motion of the entire video. Our sliding window scheme only warps the frames with respect to the temporally local motion, so we are still able to achieve similar cropping value while directly using the explicit motion loss in Eq. (3.3). The frame size of our result is also significantly greater than [LTY+16], [WYL+19] and [LYTS13], since the artifacts in their results often cause over-cropping in the final video. Since [CK20] is based on frame interpolation, their cropping score is by default equal to 1. However, [CK20] is essentially an offline method requiring multiple iterations over the entire video. In the following discussions, we will show that their distortion and stability score is much worse than

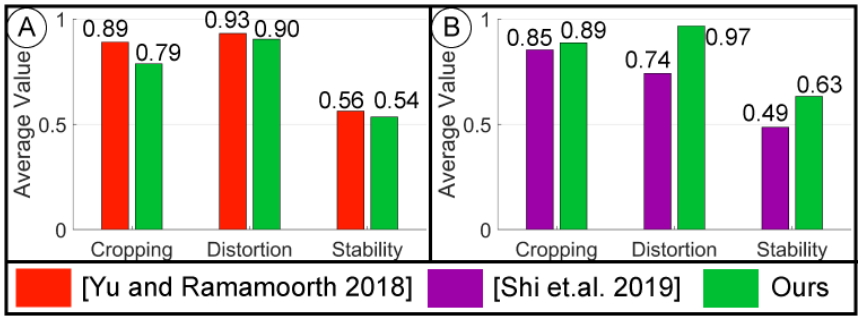


Figure 3.13: *Quantitative comparison with Ⓐ selfie video stabilization [YR19b] and Ⓑ Steadiface [STWL19] using their datasets respectively. The average values over the entire datasets are plotted. In all the three metrics, a larger value indicates a better result.*

ours.

In the second row of Fig. 3.12, we show the distortion metric. This metric measures the anisotropic scaling of the stabilized frame. A larger value indicates that the visual appearance of the result is more similar to the input video. Since we warp the frame with grid approximated moving least squares, minimal anisotropic scale was introduced to the result. The MeshFlow method [LTY+16] and bundled camera paths [LYTS13] introduces unexpected local distortion to the frame, which leads to the negative impact on the distortion value. The learning based methods [WYL+19] and [CK20] cannot generalize to selfie videos. They also produce visual artifacts that lead to even worse distortion values comparing to optimization based methods [LTY+16, LYTS13].

The bottom row of Fig. 3.12 shows the stability metric comparison. A larger stability metric indicates a more stable result. This is the most important metric for video stabilization. Comparing with the input (the yellow bar on the left of each example), our method significantly increases the stability in the result. Our method achieves a comparable result with the optimization based method [YR19b] with orders of magnitude improvement in stabilization speed. We also achieve better stability than [LTY+16, LYTS13, WYL+19, CK20], which is expected since their visual result is not satisfactory as shown in Fig. 3.11.

To further verify the performance of our method, we also test our method on the selfie

Table 3.2: Quantitative results from different network designs. In this table, *C* is the number of filters in the first layer of our network depicted in Fig. 3.6

C=32	Cropping	Distortion	Stability
No activation	0.85	0.95	0.56
Leaky ReLU	0.90	0.97	0.48
Tanh	0.87	0.97	0.50

C=64	Cropping	Distortion	Stability
No activation	0.86	0.96	0.57
Leaky ReLU	0.92	0.98	0.52
Tanh	0.85	0.96	0.52

C=128	Cropping	Distortion	Stability
No activation	0.88	0.97	0.60
Leaky ReLU	0.91	0.97	0.57
Tanh	0.89	0.96	0.52

videos provided in [YR19b] and [STWL19]. Figure 3.13 shows the average values of the three metrics above on the selfie video dataset proposed by Ⓐ [YR19b] and Ⓑ [STWL19]. Again, our result has a quantitative performance comparable with [YR19b]. Our method also performs better than [STWL19] without using the gyroscope information.

3.6.4 Network Design

Non-linear layers As discussed in Sec. 3.4.3, unlike conventional neural networks, our stabilization network does not contain non-linear activation layers. To justify this design, we added different types of activation layers after each convolutional layer in our network and compare the result with our original network design. To allow negative values in the network feature vectors, we select leaky ReLU and Tanh in our experiments. Table 3.2 shows the averaged quantitative result over the examples in Fig. 3.8 using the networks with leaky ReLU (with negative slope 0.2), tanh and no activation layers (our original design). For the stability metric

that is the most important, it can be observed that non-linear activation layers undermine the performance comparing to our original network design with the same base number of filters C. The reason for this performance degradation is that the non-linear layers break the linear input/output relationship requirement discussed in Sec. 3.4.3.

Since our network is linear, an obvious question is whether we need a convolutional network at all. In Sec. 3.4.3, we first note that the objective function L is non-linear, so a simple least squares linear solver such as in [LYTS13] cannot be used. We also discuss possible ways to formulate the system as a non-linear optimization problem. We conduct an experiment in which we optimize our loss function Eq. 3.3 directly over the feature points (warp nodes) instead of network weights. We optimize 1000 iterations using Adam optimizer [KB15] with $lr = 10^{-1}$, $\beta_1 = 0.9$ and $\beta_2 = 0.99$ for each 5-frame sliding window. Note that the runtime of this optimization is prohibitive for pratical use, since it requires an average of 20 seconds to stabilize each frame. We show the quantitative comparison of this optimization result with the result generated by our linear network in Table 3.3. Although our network is linear, it performs significantly better than direct optimization. This is expected; since the input feature points are sparsely distributed and the distribution varies frame from frame, blindly overfitting to the feature points in each sliding window will result in temporal inconsistency. Our linear network provides implicit regularization for this process since it is trained over a variety of feature point distributions. Therefore, this comparison proves that using the linear network is necessary and can produce significantly better results than optimization.

Number of filters To show the effect of the number of filters used in each layer of the network, in Table 3.2 we include the quantitative results with different numbers of filters in the input layer, i.e., $C = 32, 64, 128$. In general, the larger number of filters in the network, the better the results. This conclusion also applies to the networks with non-linear activation layers, but the effect is more significant for the leaky ReLU activated network. For the even more non-linear network with tanh layers, the performance saturates quickly with a greater number of filters C. In

Table 3.3: Linear Network vs. Direct Optimization

Methods	Cropping	Distortion	Stability
Direct Optimization	0.91	0.93	0.40
Our Linear Network	0.88	0.97	0.60

Table 3.4: Ablation Study

Ablation	Cropping	Distortion	Stability
No Foreground Detection	0.89	0.95	0.52
Full Pipeline	0.88	0.97	0.60

this chapter, we use $C = 128$ in all the experiments.

3.6.5 Ablation Study

We performed an ablation study by removing foreground mask detection stage in our pipeline. This experiment means that we are essentially using all the feature points from both foreground and background, even if the foreground feature tracking is not reliable. The stability score is significantly smaller than our full pipeline that separates the foreground and background. However, note that even without foreground mask detection, we still outperform comparison optimization based methods [LYTS13, LTY+16]. This also indicates that using the network is necessary for the video stabilization task.

3.6.6 Video Frame Size

The previously discussed results are tested with videos with frame size 832×448. Since our network only takes feature point/head vertices as the input, it is scalable with different

Table 3.5: Input Video Frame Size Comparison

Frame Sizes	Cropping	Distortion	Stability
HD (1280×720)	0.87	0.95	0.59
FHD (1920×1080)	0.87	0.96	0.58
832×448	0.88	0.97	0.60

Table 3.6: Runtime Comparison

Methods	Average stabilization time(per frame)
Ours	38ms
Selfie Video Stabilization[YR19b](Chapter 2)	4720ms
Bundled Camera Paths[LYTS13]	392ms
Steadiface[STWL19]	8ms
MeshFlow[LTY+16]	20ms
Deep Online Video Stabilization[WYL+19]	28ms
Deep Iterative Frame Interpolation[CK20]	67ms

frame sizes. We tested our network with standard video resolutions (i.e., HD 1280×720 and Full HD 1920×1080) and compare the quantitative results with the 832×448 input, shown in Table 3.5. In these experiments, we resize the frame to 832×448 for faster feature detection and foreground/face detection. In the warping stage, we rescale the feature points and the output of our network. Our network is able to handle higher resolution videos, and the result quality is similar to previously discussed results with frame size 832×448.

3.6.7 Stabilization Speed

We show the average stabilization speed of the comparison methods and our method in Table 3.6. On average, our method uses 38ms to stabilize a frame. Our code is written in Python and runs on a desktop computer with an NVIDIA 2080Ti graphics card. The break down of runtime is 3ms for foreground mask detection, 7ms for the feature detector, 3ms for KLT tracking, 16ms for face mesh detection, 5ms for stabilization network inference, less than 1ms for MLS grid approximation and 4ms for frame warping. For different video resolutions, since we rescale the feature points, the only operation for which the speed is impacted is the grid warping. However, since the grid warping is implemented on the GPU, the difference is subtle: 4ms for HD and 6ms for FHD. The overall speed is around 40ms/frame for HD and 42ms/frame for FHD. Our method is nearly two orders of magnitude faster than our previous selfie video stabilization [YR19b](Chapter 2), and nearly an order of magnitude faster than the

traditional optimization based general video stabilization [LYTS13]. Our method is also nearly two times faster than the deep frame interpolation method [CK20], since their network involves 2D convolutions. Also note that [CK20] is an offline method requiring future frames and multiple iterations through the entire video.

Although our method is slightly slower than MeshFlow [LTY+16] and deep multi-grid warping [WYL+19], we have shown in Sec. 3.6.2, Sec. 3.6.3 and supplementary video that our method produces significantly better results than theirs. Our method is also slower than Steadiface [STWL19]. However, our method is a purely software video stabilization and requires no gyroscope information, which is not available on some devices, e.g., action cameras. Our method is also able to stabilize the background in addition to the face. This makes our approach usually yield visually more stable results as we will show in our supplementary video. As we discussed earlier in Sec. 3.2, our method essentially more accurately models the frame motion than Steadiface [STWL19]. Therefore their method does not generate comparable quality as our method. Also note that our method also runs at a real-time speed without any attempt to optimize the implementation. We believe that the speed of our pipeline can be further improved by using the GPU memory sharing between feature detection/tracking and neural network operations to avoid repetitive data transferring between CPU and GPU.

3.6.8 Limitation

Our method fails if very few feature points are detected in the background, since our method requires a reasonable number of warp nodes to warp the frame. These cases include very dark environments, pure white walls and blue sky. This is a common limitation for feature tracking based methods [GKE11, LYTS13, LGW+11, GF12, LGJA09]. In our method, this can be solved by replacing the feature tracking with the optical flow algorithm with appropriate accuracy and real-time performance.

3.7 Summary

In this chapter, we proposed a real-time learning based selfie video stabilization method that stabilizes the foreground and background at the same time. Our method uses the face mesh vertices to represent the motion of the foreground and the 2D feature points as the means of background motion detection and the warp nodes of the MLS warping. We designed a two branch 1D linear convolutional neural network that directly infers the warp nodes displacement from the feature points and face vertices. We also propose a grid approximation to the dense moving least squares that enables our method to run at a real-time rate. Our method generates both visually and quantitatively better results than previous real-time general video stabilization methods and comparable results to the previous selfie video stabilization method with a speed improvement of orders of magnitude.

Our work opens up the door to high-quality real-time stabilization of selfie videos on mobile devices. Moreover, we believe that our selfie video dataset will inspire and provide a platform for a variety of graphics and vision research related to face modeling and video processing. In the future, we would explore the possibility of learning based selfie video frame completion using our proposed selfie video dataset.

This chapter is a reformatted version of the material as it appears in "Real-Time Selfie Video Stabilization," Jiyang Yu, Ravi Ramamoorthi, Keli Cheng, Michel Sarkis and Ning Bi [YRC+20]. The material has been submitted to the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2021. The dissertation author was the primary investigator and author of this paper.

Chapter 4

General Video Stabilization

4.1 Introduction

In Chapter 2 and Chapter 3, we discussed both optimization based and learning based selfie video stabilization. In this chapter, we broaden our focus to general videos. As discussed in Chapter 1, most traditional video stabilization methods rely on feature tracking and physically based constraints in the stabilization process. Examples of these methods include modeling frame motion as a full-frame 2D image transformation like Matsushita et al.[MOG+06] or a grid of local homographies like Liu et al.[LYTS13]. 3D methods seek to explore the 3D location of

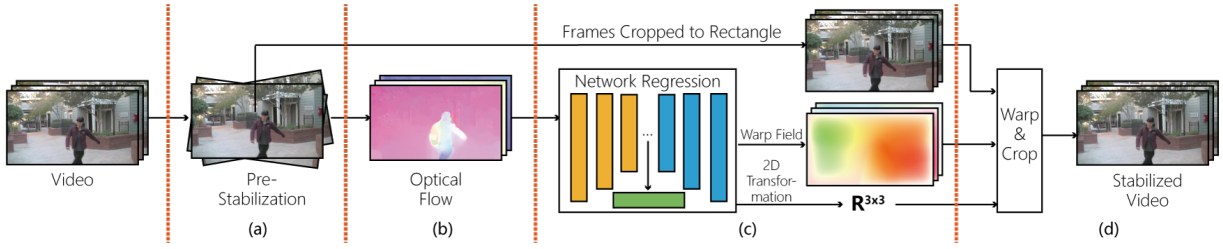


Figure 4.1: *The pipeline of our method. (a) We first pre-stabilize the video using basic 2D affine transformation. (b) Using the optical flow between consecutive frames of the pre-stabilized video, we formulate the video stabilization as the minimization of the distance between corresponding pixels. (c) We use a convolutional neural network as the optimizer to solve for a 2D affine transformation and a warp field for each frame. (d) The video frames are warped and cropped to a rectangle to produce the stabilized result.*

feature points in the scene while calculating the camera pose in 3D space. Works following this direction include Liu et al.[LGJA09], Bhat et al.[BZS+07] and Sun[Sun12]. However, all previous methods involve various heuristics. In real scenarios, complex effects (e.g. motion blur, occlusion, parallax) may unexpectedly break the assumptions in these methods and therefore produce artifacts in the results.

It would be ideal if we could make no assumptions about the physics and directly optimize the appearance change of the frames in the results so that the video is stabilized. To model the appearance change, dense optical flow fields between consecutive frames are needed. We seek to apply pixel-wise offsets to these optical flow fields, and smooth the motion of each pixel. However, modeling the frames with optical flow brings three major challenges. First, modern videos are usually high-definition. This means that the number of pixels in each frame is large. As the length of video grows, optimization quickly becomes intractable since too many pixel motions need to be solved. (For example, for a 100-frame standard 480P video(854×480), the number of motion vectors to be solved is $854*480*100 = 4.1\times10^7$). Second, as the problem size becomes large, the energy landscape of the non-convex optimization becomes complex. General gradient-based optimization algorithms may easily get stuck in local minima and yield unsatisfactory results. Third, the performance of the video stabilization is affected by the quality of optical flow. Local errors in the optical flow map will also be blindly treated as actual pixel motions, causing artifacts in the final results. Therefore, the regularization needs to be carefully designed to enforce spatial consistency and maintain robustness to the errors in the optical flow field.

Instead of trying to directly solve for the pixel-wise warp field, we propose a novel method that optimizes in the space of neural network parameters. Note that unlike standard CNN approaches, we don't use large datasets or learning of parameters a-priori. We train the CNN from scratch on a single input video. In fact, there is no traditional training in our method; the CNN is simply used as a robust way to do global optimization with a physically-based

objective function. The other important difference from traditional CNN training is that we seek to overfit the data as much as possible since our sole goal is to produce the best results for the single input video. By optimizing the parameters of the CNN rather than directly for pixels in a single test video, we make the dense warp field optimization tractable. A similar idea has been employed in image generation and restoration by Ulyanov et al.[UVL18], but is applied by us for the first time in video stabilization. This idea may be applicable in many other image and video processing applications where the physically-based problem is intractable for traditional optimization algorithms.

The pipeline of our method is shown in Fig. 4.1. We first pre-stabilize the video to reduce the frame motion (Sec. 4.3.1). We use optical flow between consecutive frames to generate dense correspondence of all pixels between the two frames. The stabilization is achieved by minimizing the distances between corresponding pixels. We seek to solve a full frame 2D image transformation and a dense warp field for each frame, so that the original frames can be warped and stabilized. We will discuss our formulation of the video stabilization problem in Sec. 4.3.

In summary, our contributions include:

Optical flow based formulation: We use the optical flow to track the actual motion of all the pixels in the video instead of pixel profiles in Liu et al.[LYTS14], which enables high robustness, universal stabilization over any part of the scene (regardless of foreground or background), and flexible non-parametric frame warping (Sec. 4.3). Our novel formulation of video stabilization leads to a large scale non-convex problem, which is addressed as discussed below with a CNN-based optimization.

Neural network based regression: We propose a new idea to transfer the video stabilization problem into a neural network based regression (Sec. 4.4). We also discuss the implementation details of the neural network regression in Sec. 4.5. We analyze the effect of different network structures on the final results, and propose a network structure that is best for video stabilization. Our network structure significantly simplifies the optimization process and generates compelling

results (Sec. 4.6).

4.2 Related Works

In this section, we summarize related works in video stabilization and neural network based regression.

4.2.1 Video Stabilization

2D Methods The 2D methods in general have low computational complexity and can be solved efficiently. However, 2D methods suffer from potential problems. First, the tracked 2D features can be unreliable due to motion blur and illumination change. Obtaining long feature tracks is also difficult in videos with significant occlusions. Liu et al.[LYTS13] tracks the 2D feature points and solves for a grid that smoothes its enclosing feature tracks. Grundmann et al.[GKE11] requires a camera path calculated from feature tracks as an initialization of the algorithm. Buehler et al.[BBM01] also requires long feature tracks and simple motion scenarios. Some methods seek to explore the relative position of feature points. Liu et al.[LGW+11] perform stabilization on the extracted eigen-trajectories. Goldstein and Fattal[GF12] utilize the epipolar geometry to maintain the relative position of feature points. Wang et al.[WLHL13] also seek to keep the relative position of feature points. These works' performances are still subject to the quality of tracked features.

Second, using a parametric motion model is usually insufficient to stabilize videos with parallax effects, since the motions of pixels in the same frame are not subject to the same homography constraint. Matsushita et al.[MOG+06] and Gleicher and Liu[GL08] treat the scene as a plane and use a full-frame homography to stabilize the video. Liu et al.[LYTS13] and our work in Chapter 2 and Chapter 3 divide the frames into grids and apply a local homography, but essentially cannot handle complex depth variation in the scene. Liu et al.[LYTS14] uses optical

flow to warp the original frames. However, the pixel profile proposed in Liu et al.[LYTS14] is very sensitive to motion discontinuities. Therefore, they still need heuristics to identify the foreground/background and carefully inpaint the regions where the motion is different from the background.

Our method is more robust than general 2D methods in terms of feature tracking, since our method tracks all the pixels and is robust to local errors in the optical flow. We also enable a non-parametric frame warping, which handles parallax effects without reconstructing the 3D structure of the scene. Although we used optical flow to synthesize stabilized frames like Liu et al.[LYTS14], our method is fundamentally different from their work. In our work, we track the actual motion of each pixel instead of the pixel profile which only collects the motion vectors at each pixel position. This makes our method robust to parallax and does not require the filling in of the motion discontinuity regions. However, our formulation results in a large scale non-convex problem which cannot be written as a simple quadratic form as in Liu et al.[LYTS14] To solve this non-trivial problem, we discuss our novel neural network based optimization routine in Sec. 4.4.

3D Methods Unlike 2D methods, 3D methods seek to explore the 3D location of feature points in the scene while calculating the camera pose in 3D space. These works in general handle parallax better than 2D methods, since the motion is physically analyzed in actual 3D space. In these works, the camera path is smoothed and the 3D feature points are reprojected to new camera positions in order to guide the warping[LGJA09, Sun12] or the methods use image-based rendering[BZS$^+$07] to synthesize a frame from the original frames. However, the 3D methods suffer from robustness and complexity issues in Structure from Motion.

There are also video stabilization methods that require specific hardware information or focus on video captured with a specific camera. Sun[Sun12] requires a depth camera for video stabilization. Smith et al.[SZJA09] requires a light field camera. Karpenko et al.[KJBL11] uses gyroscope information to help in stabilizing the video. Kopf[Kop16] focuses on videos captured with a 360° camera. Some of these works show strong results, but have limited application since

most videos do not include the extra information required in these algorithms.

Deep Learning Based Methods Some recent works seek to use a pre-trained network in video stabilization tasks. Wang et.al.[WYL+19] train a two-branch Siamese network and try to directly predict the homography transformation from the current frame and previous stabilized frames. Xu et.al.[XHW+18] also try to use spatial transformer networks (STN) to predict the affine transformation. Moreover, they use an adversarial network to directly generate previous stabilized frames instead of the real frames in Wang et.al.[WYL+19]. Although these works utilize pre-trained CNNs, they still use a simple full-frame transformation as the motion model, which cannot handle parallax effects. Lin et.al.[LJL+17] proposed a mesh deformation algorithm by enforcing the photo consistency between images. Their semi-dense photometric alignment provides better robustness compared to methods using feature points. However, the photometric based metric can be unreliable in homogeneous regions, object boundary and occluded regions. These regions are also challenging for optical flow algorithms, but as we will discuss in Sec. 4.3, our regularization can help avoid visual artifacts in these regions.

4.2.2 Neural Network Regression

Deep convolutional neural networks have been used in various image/video processing tasks. Such applications include image super-resolution[DLHT14, LHAY17, LTH+17, TYL17], image denoising[Lef16], HDR reconstruction[EKD+17], panorama video loop generation[HLSH18] and video interpolation[JSJ+18]. Some of these methods are designed for processing a single image/video, but their networks are essentially trained on a large dataset of images/videos. Unlike traditional deep learning, our method treats the network purely as an optimizer over one single input video. We train the network from scratch for each specific video, and try to overfit and obtain the best result for the input.

Ulyanov et al.[UVL18] recently proposed that a randomly-initialized neural network can be used in learning image priors on a single image. The idea is that the variables of a typical

optimization task can be replaced by the output of a neural network. The neural network is randomly initialized and trained on a single image to minimize the loss function designed for a specific task, e.g. denoising, superresolution, and inpainting. This enables the optimization in the neural network parameter space instead of image space, while dramatically improving the result. In this work, we expand this idea into video processing, which is more difficult than the single image processing scenario in terms of the problem formulation and complexity. By transferring the video stabilization problem into neural network parameter space, we can easily solve the large scale problem, which is difficult to solve using traditional optimization methods. Moreover, we further analyze the effects of different types of architectures on the final optimization result. We will discuss this idea in detail in Sec. 4.4.

4.3 Optical Flow Based Objective Function

It is well-known that videos usually have multiple factors that cause difficulties for video stabilization algorithms, e.g. lens distortion, motion blur, dynamic objects, parallax, low-illumination etc. These effects can be individually modeled using hand-crafted physically based models. However, in real-world videos, effects usually couple with each other, making algorithms specifically designed for one single effect fail in other cases. Instead of trying to physically model these complex effects, we treat the video stabilization task as a pure 2D image processing problem. In this chapter, we seek to minimize the appearance change among video frames.

4.3.1 Pre-Stabilization

Although we have considered the inaccuracy of the original optical flow, a pre-stabilization is still necessary to reduce the motion and improve the quality of the original optical flow. Moreover, for large motion videos, a large number of the boundary pixels have no correspondence in the next frame. This results in artifacts in the boundary region of the output warp field, since

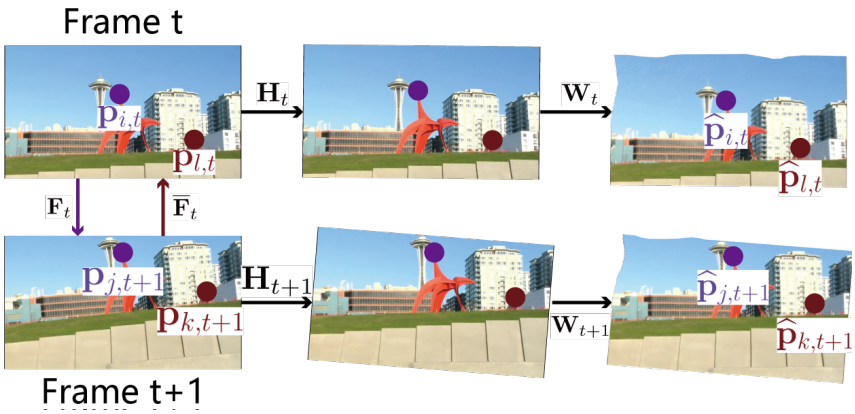


Figure 4.2: *The objective function of our method. The computed bidirectional original optical flow provides correspondence among all pixels in two consecutive frames. In this figure,* $\mathbf{p}_{i,t}$ *is associated with* $\mathbf{p}_{j,t+1}$ *by* $\mathbf{F}_t$ *and* $\mathbf{p}_{k,t+1}$ *is associated with* $\mathbf{p}_{l,t}$ *by* $\overline{\mathbf{F}}_t$*. Our goal is to solve a 2D affine transformation* $\mathbf{H}$ *and a warp field* $\mathbf{W}$ *to minimize the distance between the associated pixels.*

these pixels can be warped freely without any constraint from neighboring frames. Therefore, our method pre-stabilizes the video before stabilizing the video with the process discribed in the rest of this section.

In the pre-stabilization phase, we first use KLT[ST94] to track minimum eigenvalue[ST94] feature points over all the frames. Denote a feature point at time t as $\mathbf{f}_{i,t}$ and its correspondence in $t+1$ as $\mathbf{f}_{i,t+1}$ respectively. We solve for a per-frame 2D affine transformation matrix K_t such that the integral of squared second derivative is minimized:

$$E(K) = \sum_{i,t} \|K_t f_{i,t} - K_{t+1} f_{i,t+1}\| \tag{4.1}$$

The solved K_t are used to transform the frames of the input video, and the result is cropped to a rectangle as the output of the pre-stabilization phase. The output of this pre-stabilization is used as the input for the rest of this section.

Table 4.1: Notations used throughout this chapter.

Notation	Meaning	Size
w	Frame width	1
h	Frame height	1
t	Frame index(time)	1
T	Total number of frames	1
$\mathbf{I}_t$	Input RGB frame at t	$w \times h \times 3$
x, y	Pixel coordinate	1
i, j, k, l	Pixel ID	1
$\mathbf{p}_{i,t}$	Spatial location of pixel i at t	2×1
$\mathbf{p}_{i,t}^h$	Homogeneous version of $\mathbf{p}_{i,t}$	3×1
$\widehat{\mathbf{p}}_{i,t}$	Warped $\mathbf{p}_{i,t}$	2×1
$\mathbf{P}_t$	Coordinates of all pixel in frame I_t	$wh \times 2$
$\mathbf{S}_t$	The coordinates of four corner pixels	4×2
$\mathbf{D}$	Weight of all pixels in the 2D interpolation representation w.r.t. $\mathbf{S}$	$wh \times 4$
$\mathbf{F}_t$	Optical flow from frame I_t to I_{t+1}	$w \times h \times 2$
$\overline{\mathbf{F}}_t$	Optical flow from frame I_{t+1} to I_t	$w \times h \times 2$
$\mathbf{H}_t$	2D affine transformation	2×3
$\mathbf{W}_t$	2D warp field	$w \times h \times 2$
θ	Neural network parameters	
$\mathbf{G}(\theta)$	Neural network as a function of θ	

4.3.2 Optical Flow Objective Function

In Sec. 4.3.1, we first perform a pre-stabilization step, where we track sparse feature points and preliminarily stabilize the video. To model the appearance change, the next step of our approach is to calculate original optical flow between consecutive video frames and find frame transformations to minimize the motion of pixels. To make the following discussion clear, we define the notations in Table 4.1.

To simplify the notation, we unroll the pixel coordinate x, y to a single pixel ID i. Denote the original optical flow from t to $t+1$ as $\mathbf{F}_t$, a two-channel image that encodes the shift of all the pixels in frame $\mathbf{I}_t$ to frame $\mathbf{I}_{t+1}$. For example, denote the position of pixel i in frame $\mathbf{I}_t$ as $\mathbf{p}_{i,t}$. Its corresponding pixel in frame $\mathbf{I}_{t+1}$ can be represented as

$$\mathbf{p}_{j,t+1} = \mathbf{p}_{i,t} + \mathbf{F}_t(\mathbf{p}_{i,t}) \tag{4.2}$$

Similarly, a backward optical flow can be computed and maps the pixels in frame $\mathbf{I}_{t+1}$ to frame $\mathbf{I}_t$

$$\mathbf{p}_{l,t} = \mathbf{p}_{k,t+1} + \overline{\mathbf{F}}_t(\mathbf{p}_{k,t+1}) \tag{4.3}$$

We illustrate our approach in Fig. 4.2. Our goal is to warp each original frame so that the output frames are stabilized. The warping operation consists of two components: a 2D affine transformation $\mathbf{H}_t$ and a per-pixel warp field $\mathbf{W}_t$. Therefore, the warped pixel i and l in frame $\mathbf{I}_t$ can be represented as

$$\begin{aligned} \widehat{\mathbf{p}}_{i,t} &= \mathbf{H}_t \mathbf{p}^h_{i,t} + \mathbf{W}_t(\mathbf{p}_{i,t}) \\ \widehat{\mathbf{p}}_{l,t} &= \mathbf{H}_t \mathbf{p}^h_{l,t} + \mathbf{W}_t(\mathbf{p}_{l,t}) \end{aligned} \tag{4.4}$$

where $\mathbf{p}^h_{i,t}$ and $\mathbf{p}^h_{l,t}$ stands for the homogeneous representation of $\mathbf{p}_{i,t}$ and $\mathbf{p}_{l,t}$. Similiarly, its warped correspondence in frame I_{t+1} is

$$\begin{aligned} \widehat{\mathbf{p}}_{j,t+1} &= \mathbf{H}_{t+1} \mathbf{p}^h_{j,t+1} + \mathbf{W}_{t+1}(\mathbf{p}_{j,t+1}) \\ \widehat{\mathbf{p}}_{k,t+1} &= \mathbf{H}_{t+1} \mathbf{p}^h_{k,t+1} + \mathbf{W}_{t+1}(\mathbf{p}_{k,t+1}) \end{aligned} \tag{4.5}$$

The objective is to minimize the Euclidean distance between the warped pixel positions:

$$\begin{aligned} &E_o(\mathbf{W}, \mathbf{H}) = \frac{1}{wh(T-1)} \sum_{t=1}^{T-1} \\ &(\sum_{i=1}^{wh} \left\| \widehat{\mathbf{p}}_{i,t} - \widehat{\mathbf{p}}_{j,t+1} \right\|^2 + \sum_{k=1}^{wh} \left\| \widehat{\mathbf{p}}_{l,t} - \widehat{\mathbf{p}}_{k,t+1} \right\|^2) \end{aligned} \tag{4.6}$$

where wh is the total number of pixels in a frame and T is the total number of frames. Note that the mapping from $\mathbf{p}_{i/l,t}$ to $\mathbf{p}_{j/k,t+1}$ is 1-to-1, so we only need to average over i and k.

4.3.3 Regularization

Due to the complexity of scenes, the original optical flow could be inaccurate in some regions. Moreover, objects in the scene might be moving regardless of the motion of the camera. Blindly optimizing the objective function (4.6) could introduce artifacts. An example of these

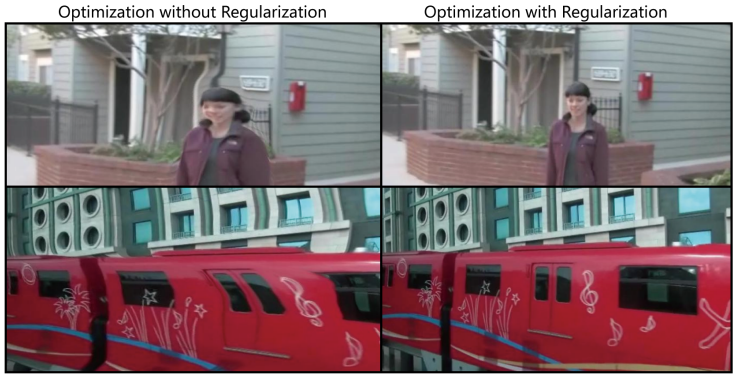


Figure 4.3: *Video stabilization results using regularization vs. no regularization. (Left) Due to the moving object in the scene and the inaccurate original optical flow, unexpected artifacts are introduced if the video is stabilized purely according to optical flow. (Right) Applying proper regularization helps reduce the visual artifacts.*

artifacts is shown in Fig. 4.3. Therefore, we seek to enforce the local continuity of the output warp field $\mathbf{W}_t$.

The four corner pixels define a rectangular region, in which each pixel position $\mathbf{p}_{i,t}$ can be represented by a linear interpolation of the coordinates of the four corner pixels: $\mathbf{P}_t = \mathbf{D}\mathbf{S}_t$, where each row of $\mathbf{P}_t$ is the coordinate of pixels, each row of $\mathbf{D}$ is the 2D interpolation weight, and each row of $\mathbf{S}_t$ is the 2D coordinates of the four corners. Note that moving the corner position correspondingly changes all the pixel locations:

$$\Delta\mathbf{P}_t = \mathbf{D}\Delta\mathbf{S}_t$$

A warp field obeying this linear warping rule should satisfy:

$$\|\mathbf{W}_t - \mathbf{D}\Delta\mathbf{S}_t\|_2 = 0 \tag{4.7}$$

However, our output warp field $\mathbf{W}_t$ will not exactly be a linear warping. Our goal is to keep the term in (4.7) as small as possible. The least squares representation of $\Delta\mathbf{S}_t$ is:

$$\Delta\mathbf{S}_t = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{W}_t$$

This estimation of $\Delta\mathbf{S}_t$ leads to the error of:

$$E_r(\mathbf{W}) = \mathbf{W}_t - \mathbf{D}(\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{W}_t \tag{4.8}$$

We use this error as a constraint to enforce the output warp field close to a linear warp field. Note that this formulation allows us to control the linear warping constraint at a pixel-level, simply by changing $\mathbf{D}$. For example, in our experiment, we cover each frame with a 20x20 grid and fill $\mathbf{D}$ with the weight of each pixel in its enclosing grid cell.

Moreover, the original optical flow $\mathbf{F}_t$ is less reliable for regions with large motions. To take this into consideration, we tend to increase the regularization value (4.8) for large motion regions to obtain $\mathbf{W}_t$ with fewer discontinuities; on the other hand, for small motion regions, we tend to trust the optical flow and decrease the regularization value. The measurement of motion scale can be estimated using the pixel motion obtained from the original optical flow:

$$E_p = \mathbf{F}_t^2 + \overline{\mathbf{F}}_t^2 \tag{4.9}$$

4.3.4 Final Objective Function

Combining (4.6), (4.8) and (4.9), our optimization problem can be written as:

$$\min_{\mathbf{W},\mathbf{H}} E_o(\mathbf{W},\mathbf{H}) + \lambda \left\| E_p \cdot E_r(\mathbf{W}) \right\|_1 \tag{4.10}$$

where λ is a hyperparameter controlling the amount of regularization in general. Since the magnitude of original optical flow E_p has the same size as the regularization E_r, we use it as a pixel-wise weight to E_r. Note that to encourage sparsity in the warp field and avoid over compensation to erroneous regions in the original optical flow, we use L_1 norm for this regularization.

Discussion Our formulation directly models the motion of every pixel in a video using dense optical flow. Unlike previous works that use various heuristics, our method is based on the first principles that we should stabilize what we finally perceive. The most similiar idea is the SteadyFlow proposed by Liu et al.[LYTS14]. However, they only collect the motion vector on fixed pixels. The motion vectors on a single pixel correspond to the motion of different locations in the scene. The accumulation of these vectors does not match the true motion of the camera. Our formulation is novel since we physically model the motion of every visible point in the scene. This leads to a more difficult optimization problem, as we will discuss in Sec. 4.4.

4.4 Convolutional Neural Network Regression

Note that the unknowns in (4.10) are a per-frame optical flow field $\mathbf{W}_t$ and a per-frame 2D affine transformation $\mathbf{H}_t$. For a 300-frame video clip with a standard 480p resolution, the total number of unknown motion vectors is approximately 123 million. Directly optimizing a problem of this size is typically prohibitive due to the computation cost and limited memory. Moreover, the optimization will be difficult due to the complex high-dimensional energy landscape with a large number of local minima.

Our main idea to solve this problem is to search for the answer in the neural network parameter space instead of in the problem space. In fact, we are using the neural network as an optimizer. Our method is different from traditional learning on large datasets. There is no training set, and the network weights are directly optimized on the input video with the objective function in (4.10). Using a network makes this non-convex high-dimensional optimization problem practical, enabling us to directly use a robust optical-flow based stabilization formulation.

To our knowledge, our method is the first work that uses this idea in video stabilization tasks. Another insight is that although the optical flow field is represented pixel-wise, it is spatially smooth for real-world scenes. Therefore, our warp field $\{\mathbf{W}, \mathbf{H}\}$ can be described well

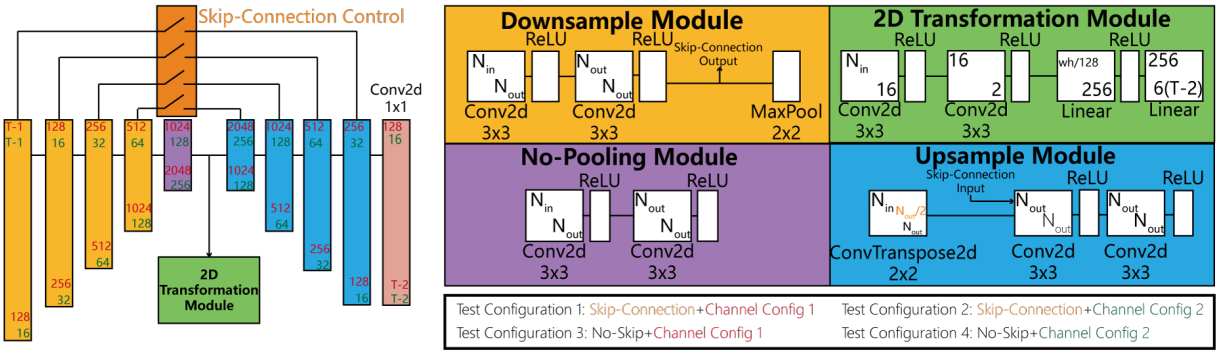


Figure 4.4: *Our network structure. The overall structure is shown on the left. The details of each type of module are shown on the right. The numbers shown on the upper-left/lower-right corner of each module represent the number of input/output channels. The numbers in red/green represent two different channel configurations of the network. The orange box represents optional skip-connections between downsample modules and upsample modules. Combining the channel configurations and skip-connection options, a total of 4 different configurations are used in our experiment (listed on the lower-right of the figure).*

by a parameterized function. However, given the complexity of this process, a complex and differentiable parameterized function needs to be designed. Instead of hand-crafting this function, we select the convolutional neural network as an ideal out-of-the-box solution for this task.

Denote the neural network as a function $\mathbf{G}(\theta)$ where θ represents the parameters of the network. We seek a set of network weights so that the output of the network is the desired warp field $\{\mathbf{W},\mathbf{H}\} = \mathbf{G}(\theta)$. Therefore, the optimization problem (4.10) can be reformulated as:

$$\min_{\theta} E_o(\mathbf{G}(\theta)) + \lambda \left\| E_p \cdot E_r(\mathbf{G}(\theta)) \right\|_1 \tag{4.11}$$

The goal becomes searching for the parameters θ by training the network on a single video clip. Note that since our objective function consists of simple linear and quadratic functions of $\{\mathbf{W},\mathbf{H}\}$, (4.11) is differentiable with respect to network parameters θ.

Our network structure is shown in Fig. 4.4. The input of the network is a set of $T-1$ original optical flow fields $\mathbf{F}$ computed from input video frames. The frames of optical flow fields are sent in as different channels. The input is encoded by 5 layers of downsample modules. Each downsample module downsamples the frame size by 2 but doubles the number of channels,

except the last one that only doubles the number of channels. The decoder consists of 4 layers of upsample modules followed by an output convolutional layer with kernel size 1×1. The output of the decoder is the desired warp field $\mathbf{W}$. In addition, we fed the encoded information into a 2D transformation module consisting of two convolutional layers and two linear layers. This module produces the desired 2D affine transformation matrix $\mathbf{H}$. The number of output $\{\mathbf{W}, \mathbf{H}\}$ pairs is $T-2$. We will explain why we have $T-1$ input channels and $T-2$ output channels, and discuss the selection of T in Sec. 4.5. Since we only input the optical flow of a single video and try to optimize the network parameters, we seek to overfit the single input video as much as possible. Therefore we avoid inserting dropout layers and any regularization on network weights.

As noted in Fig. 4.4, we have two different channel configurations. The channel configuration 1 requires more network parameters, while the channel configuration 2 leads to a simpler network. In addition, the network can optionally include skip connections. Combining these choices, we have four different network configurations in this chapter. We will compare the performance of these configurations in Sec. 4.6. Specifically, we will show the relation between the network configuration and the regression error (4.10) in Fig. 4.12 and discuss how the network configuration will affect the optimization performance. We will also show the network configuration's effect on the final stabilization result in Fig. 4.12.

4.5 Implementation Details

Sliding Window We now explain the details about why we have $T-1$ optical flow fields as the network input and $T-2$ warp fields as the network output. Since the input video may have different lengths, we stabilize a video using a sliding window approach. The process is demonstrated in Fig. 4.5, where an example with $T=6$ is shown. In this case, the window covers $T=6$ video frames and $T-1=5$ original optical flow frames. Note that since the input of our network is the original optical flow, the number of input channels is $T-1$. The desired number of

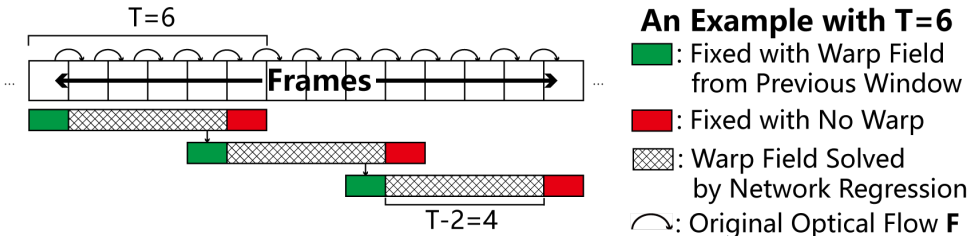


Figure 4.5: *The sliding window example with window size $T = 6$. For each window, the warp field of the first frame is fixed as computed from the previous window. The last frame is fixed so that it is not warped. We optimize the objective function discussed in Sec. 4.3 for the entire length T window and solve for $T-2$ warp fields.*

estimated $\{\mathbf{W},\mathbf{H}\}$ pairs should be 6, which can warp each frame and generate stabilized frames. However, to enforce temporal consistency, we make the windows overlap by two frames and fix the warp field of the first frame. We also fix the last frame to retain the global motion of the original video. The warp field of the first frame is copied from the estimation of the previous window. The warp field of the last frame is fixed to $\mathbf{W} = \mathbf{0}, \mathbf{H} = \mathbf{I}$ for the current window, but will be re-optimized as the second frame of the next window. The last frame of the last window remains unwarped. Therefore, for each window, we have $T-2 = 4$ pairs of warp fields $\{\mathbf{W},\mathbf{H}\}$ as the network output.

Selection of Window Size It is clear that the selection of T will affect the complexity of the optimization problem. The more frames we want to stabilize at the same time, the more complex the energy landscape will be. In Fig. 4.6, we show the error descent of optimization using T from 10 to 80 frames with a step size of 10. The y axis represents the percentage of the error of current iteration with respect to the initial error. Each curve is the averaged result for all segments of our examples in Sec. 4.6. It shows that in smaller T cases, the optimization converges faster but yields higher error after convergence. This is because in a shorter video segment, we have fewer degrees of freedom in the warp field. Although a larger window size leads to better error performance, more memory and iterations are required to stabilize a video segment. Taking all these into consideration, we select $T = 60$ in our experiments.

Miscellaneous We use the Liu[Liu09] to compute the original bidirectional optical flow.

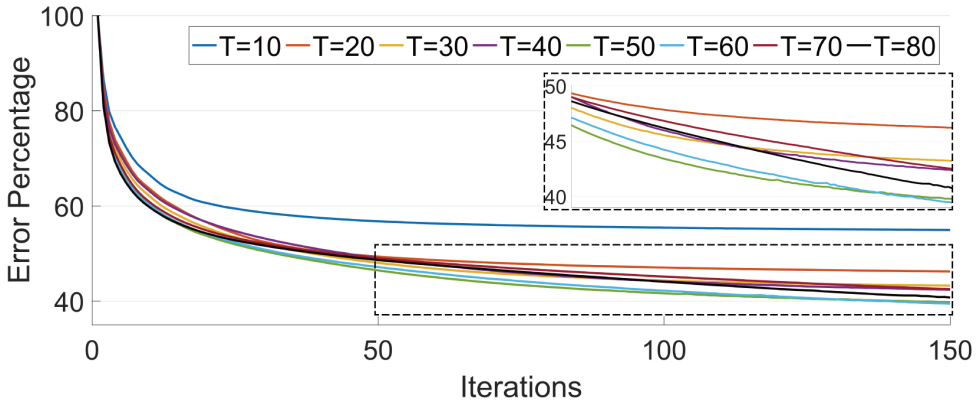
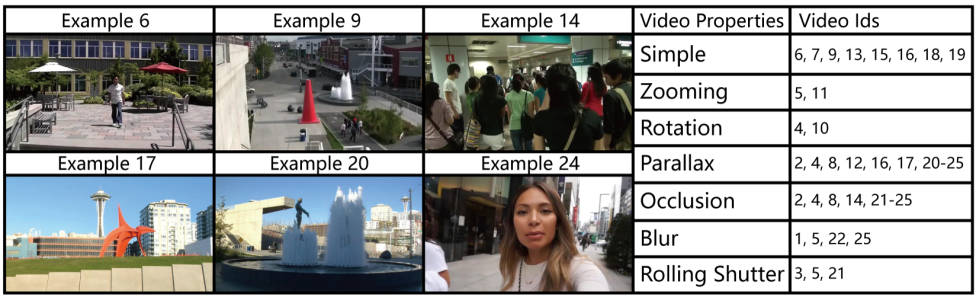


Figure 4.6: *Comparison of results stabilized with window size $T = 10$ to $T = 80$. The inset shows a zoom-in of the region circled by the dotted box. Smaller window cases converge faster but result in higher error. Larger window cases yield lower error but converge slower.*

Before computing the original optical flow, we pre-stabilize the video to eliminate large motions. The reason for the pre-stabilization stage includes two aspects: the quality of optical flow is undermined by large motions; the pixels in the boundary regions do not have correspondence in their neighboring frame, and this effect is significant in large motion cases. The details about pre-stabilization are discussed in Sec. 4.3.1. The regularization value λ in (4.10) is set to be 0.5 for all the examples shown in the supplementary video. The optimizer we used is Adam[KB15] with $\beta_1 = 0.5$, $\beta_2 = 0.59$ and a learning rate of 10^{-4}. We optimize for 150 iterations for each $T = 60$ window.



Video Properties	Video Ids
Simple	6, 7, 9, 13, 15, 16, 18, 19
Zooming	5, 11
Rotation	4, 10
Parallax	2, 4, 8, 12, 16, 17, 20-25
Occlusion	2, 4, 8, 14, 21-25
Blur	1, 5, 22, 25
Rolling Shutter	3, 5, 21

Figure 4.7: *Example stills of our examples. The example numbers are labeled above the frames. In the right table, we also summarize their properties that have significant effects on video stabilization algorithms.*

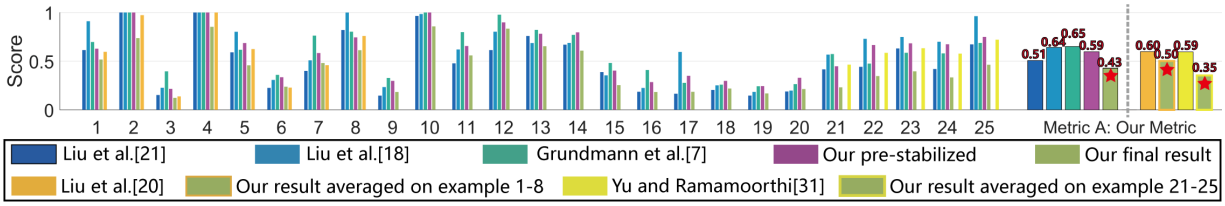


Figure 4.8: *Quantitative comparison using our metric. In this figure, a lower bar indicates a better result. The result is normalized with the score of the input video. In the left figure, results that have a normalized score greater than one are clamped in this figure. The right figure shows the averaged results over the entire 25 examples. On the rightmost part of the figure, we only compare to Liu et al.[LYTS14] using their data (examples 1-8) since we don't have their implementation. We compare to our selfie video stabilization method in Chapter 2 only on selfie videos (examples 21-25). We mark our results with the red stars. The exact scores are marked on top of each bar.*

4.6 Results

In Fig. 4.7, we show example frames of the video clips used in this chapter. In order to collect a large enough set of examples for comparison, we combined datasets from many previous papers. In our dataset, numbers 1-8 are taken from Steadyflow[LYTS14], numbers 9-15 are taken from Liu et al.[LYTS13], numbers 16-20 are taken from Liu et al.[LGJA09], and numbers 21-25 are taken from Yu and Ramamoorthi[YR19b](Chapter 2). We also summarize their properties that have significant effects on video stabilization algorithms in the right table in Fig. 4.7.

We use five metrics to evaluate the quality of the results. Our result is generated with Config 1 mentioned in Fig. 4.4. We will further discuss the effect of network configuration later in this section.

Quantitative Results Using our Objective Function: In Fig. 4.8, we show quantitative comparison of the result quality of the input video, our result and Steadyflow[LYTS14], Grundmann et al.[GKE11], Liu et al.[LGJA09], Liu et al.[LYTS13] and Yu and Ramamoorthi[YR19b](Chapter 2). Metric A is our metric, which is defined as the accumulated optical flow over the entire video:

$$\frac{1}{wh(T-1)}\sum_{t=1}^{T-1}\sum_{x=1}^{h}\sum_{y=1}^{w}\left(\|\mathbf{F}_t(x,y)\|_2+\left\|\overline{\mathbf{F}}_t(x,y)\right\|_2\right)$$

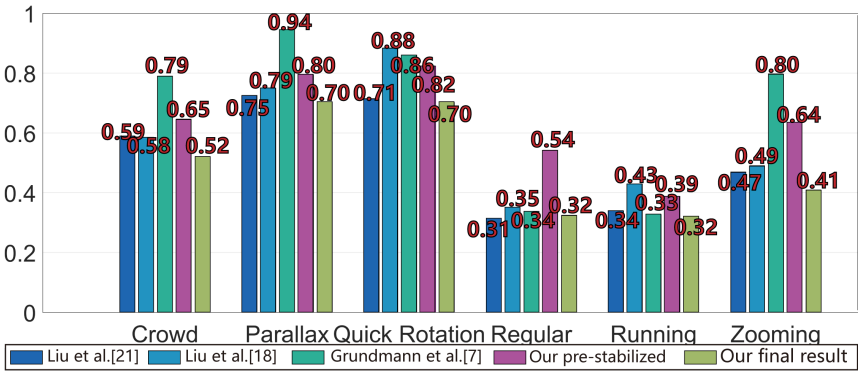


Figure 4.9: *Metric A evaluation over the results for the NUS dataset. We randomly select 5 videos from each category and average the results. Note that our network optimization significantly improves the pre-stabilization results. Our method is also better than comparison methods in challenging categories like Crowd, Parallax and Running.*

which evaluates the appearance change between consecutive frames in the results. The essence of this metric is similar to our objective function (4.6). However, the metric is different from (4.6): the optical flow it uses is computed from the resulting video. Note that to compare videos with different frame sizes, we normalize the optical flow by its frame size. A smaller score indicates a better result in metric A. To show the amount of stability improvement, all the scores are normalized by the score of the input video.

In our work, we directly tried to minimize the overall appearance change. Therefore, our method achieves the best result on average and performs better than comparison methods under this metric. Note the benefit gained by using our CNN based optimization framework, comparing to the pre-stabilized result. We also perform significantly better in example 8, which contains large foreground occlusion and is claimed as a limitation case in SteadyFlow[LYTS14]. For selfie videos (example 21-25) in which large occlusion exists, we are also able to achieve smoother appearance change. Our method obtains a larger score comparing to Liu et al.[LYTS13] in example 12, but their result contains large visual distortion as we will discuss later in this section. For some examples (6, 7, 9, 11, 17, 20), our result has a slightly higher score than the comparison methods, but there are no visible quality differences with the other methods.

In Fig. 4.9, we also compare our method with the other video stabilization methods over the commonly used NUS dataset[LYTS13]. We randomly select 5 videos from each category and average the Metric A of the results. Our method performs better on this more general video stabilization dataset.

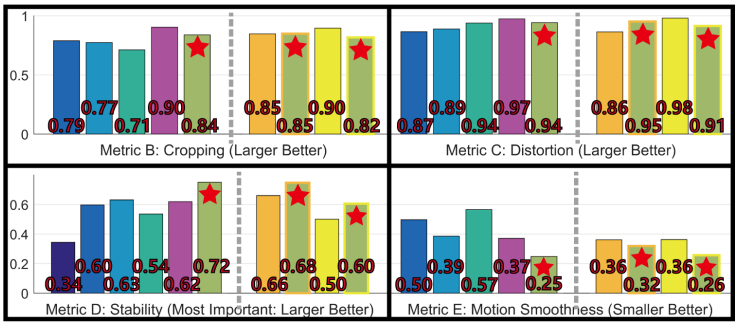


Figure 4.10: *Quantitative comparison using metrics proposed by Liu et al.[LYTS13] and Yu and Ramamoorthi[YR19b](Chapter 2). Metric B measures the cropping ratio compared to the input video, metric C measures the global distortion, metric D measures the frequency domain stability and metric E measures the motion smoothness. We mark our results with the red stars. The exact values are marked at the bottom of each bar. Metric D is considered as the most important metric.*

Quantitative Results Using Other Metrics: The metrics B, C and D were proposed by Liu et al.[LYTS13], which evaluates the results' cropping ratio, global distortion and frequency domain stability. In metrics B, C and D, a higher value indicates a better result. Metric E was proposed in Chapter 2, which evaluates the smoothness of the frame motion in the result. In metric E, a smaller value indicates a better result. In Fig. 4.10, we show comparison of averaged score over the entire 25 examples. The full comparison on each individual video is provided in the supplementary material.

In metrics B and C, since we warp the pre-stabilized video using the warp field and crop to a rectangle, we expect the final result to be slightly worse than the pre-stabilized result in terms of cropping and distortion. However, we are still achieving better results than comparison methods in metric B (cropping) and comparable result in metric C (distortion). In terms of metric D (stability), which is the most important aspect of video stabilization, we outperform the

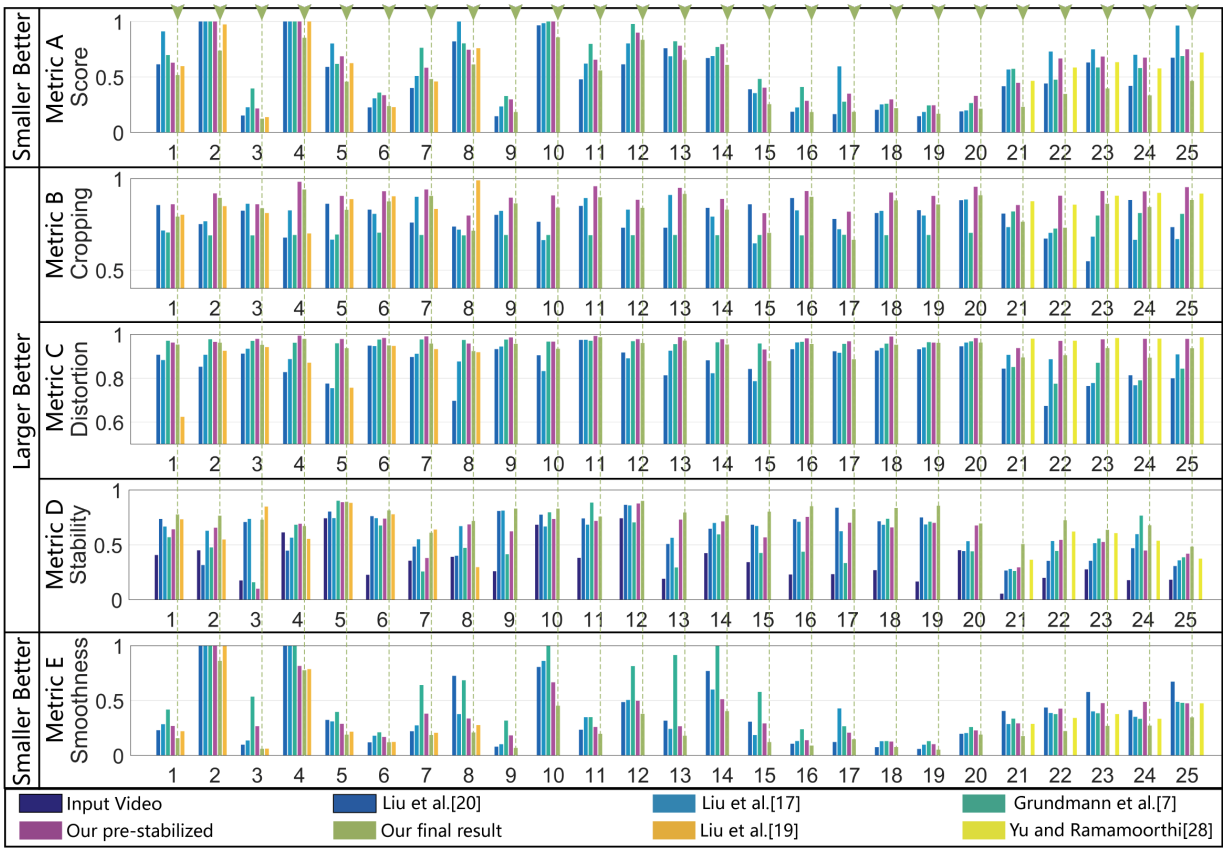


Figure 4.11: *Complete quantitative comparison using our metric (Metric A), Liu et al.[LYTS13] metric (Metric B, C and D), Yu and Ramamoorthi[YR19b](Chapter 2) metric (Metric E). In metric A and E, a lower bar indicates a better result. In metric B, C and D, a higher bar indicates a better result. The result is normalized with the score of the input video for Metric A and E. Results that have a normalized score greater than one are clamped in this figure. Our method is marked with the dotted lines. The average values for these metrics are shown in Fig. 8 and 9.*

comparison methods on average. For metric E (motion smoothness), our method also performs significantly better. Note that comparing to the method specifically designed for selfie video in Chapter 2, we are also able to achieve both better metric D (frequency stability) and metric E (motion smoothness) without explicitly modeling the human face. The complete quantitative comparison using all metrics on each individual example is shown in Fig. 4.11.

Visual Comparisons in Video: Besides the quantitative metrics, we also show visual comparison in the supplementary video. For videos with large occlusion (example 2, 4, 8, 14,

21-25), feature track based 2D methods[GKE11, LYTS13] fail due to the difficulty in obtaining long feature tracks. They also produce artifacts in videos coupled with other effects: extreme motion (example 15), motion blur (example 5), rolling shutter (example 3 and 5) and parallax (example 12). 3D methods[LGJA09] also cannot produce satisfactory results since structure from motion is not suitable for dynamic scenes in general. The optical flow based method[LYTS14] failed in challenging cases like example 8, since its heuristic on motion completion cannot handle large foreground occlusions. Our method is more robust in these cases. We do not explicitly handle the motion discontinuity, but resort to the continuity regularization (4.8) and (4.9) and make it part of the optimization. We are able to handle this complex optimization problem thanks to optimizing the neural network parameters instead of the warp field itself.

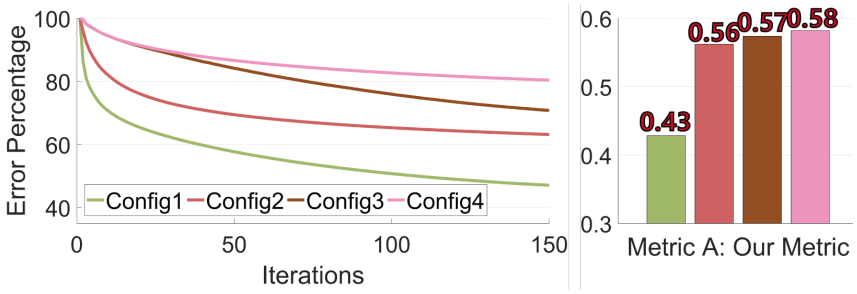


Figure 4.12: *Effect of network structures. The figure on the left shows the regression error using different network configurations shown in Fig. 4.4. The right figure shows the effect on the final result using our metric.*

Evaluation of Network Configurations: Now we discuss the effect of neural network configuration on the video stabilization result. As noted in Fig. 4.4, we have 4 different network configurations in our experiment. In the left part of Fig. 4.12, we compare the regression errors defined in (4.10) over 150 optimization iterations using these configurations. Each curve represents the average regression error on a single video segment with length $T = 60$. Networks with more channels (Config 1 and 3) can achieve lower error than networks with the same structure but fewer channels (Config 2 and 4). Fig. 4.12 also shows that networks with more complex structure (Config 1 and 2) can descend to lower error than networks with the same channels but with simpler structure (Config 3 and 4). We also compare the quantitative evaluation of the

result videos using different network configurations in the right part of Fig. 4.12. The values are averaged over the 25 example videos in this chapter. The simple networks (Config 2, 3 and 4) cannot achieve equal quality results as the most complex network (Config 1), and the difference in quality among these simple networks are less significant. This proves that the precise network architecture is important in the our case, and we find the Config 1 network is the best for video stabilization.

4.7 Summary

In this chapter, we proposed a new video stabilization formulation based on first principles. This formulation leads to a large scale non-convex optimization problem that previous works tried to avoid by proposing various heuristics. We also proposed a novel CNN based optimization routine for this problem, which does not require a large dataset and is re-trained on each single video. Our method can be applied on any video regardless of the complexity of the scene.

The limitation of our method is the computation time. Our method is an offline method which requires about 30min to stabilize a 300-frame video on a GTX1080Ti graphics card. Since we do not focus on computation time in this chapter, we believe the algorithm can be further speeded up, for example, using unidirectional optical flow and/or other network structures and channel configurations.

Our work is the first that explores the possibility of applying CNN techniques to video stabilization. An interesting future work would be a universal pre-trained neural network based on a large video dataset, followed by a fast video-specific fine-training pass. We have made preliminary efforts in this direction, but the training on a dataset of video segments does not yet converge. However, we believe that a CNN can be trained with a slight modification of our algorithm, and significantly speed up the video stabilization process.

This chapter is a reformatted version of the material as it appears in “Robust Video

Stabilization by Optimization in CNN Weight Space," Jiyang Yu and Ravi Ramamoorthi in the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2019 [YR19a]. The dissertation author was the primary investigator and author of this paper.

Chapter 5

Learning-based General Video Stabilization

5.1 Introduction

For general videos, we have proposed a method that compensates input video optical flow using dense warp field in Chapter 4. We also analyzed that it is impractical to solve the optimization problem directly due to the problem size. Therefore, we used a CNN to generate a dense warp field and optimize the problem over the CNN weights. An obvious question to ask is if we can train a neural network for arbitrary general video stabilization instead of optimizing for each individual video. In this chapter, we propose a fully automatic learning based approach for general video stabilization. Similar to Chapter 4, we use the optical flow to understand the frame motions. However, we observe that the global linear warp field constraint (Eqn. 4.8) is weak and cannot provide enough generality if we seek to train a neural network for arbitrary general video. The key problem is that the optical flow essentially only provides a pseudo correspondence between two frames. At occlusion boundaries, pixels can either appear in the next frame or be occluded in the next frame. The optical flow is also inaccurate in regions lacking texture. Using

the optical flow as the reference can lead to unexpected artifacts in the output warp fields. On the other hand, enforcing a global linear warping constraint regularizes the training too much, leading to the lack of stability in the result. In this chapter, we propose a pipeline that is specifically designed to handle the inaccuracy in the optical flow. We are the first method that uses the optical flow principal components[WB15] in video stabilization instead of hand-crafted spatial smoothness constraints. We will discuss the details of our pipeline in Sec. 5.3.

The core of our algorithm is a deep neural network that takes the optical flow as the input and directly outputs the warp fields. Our neural network based method overcomes the major drawback of optical flow based methods: the computational complexity. Both SteadyFlow[LYTS14] and our method in Chapter 4 use optimization to minimize an objective function with a significant amount of unknowns(the motion vectors in the warp fields). The optimization process must be performed for each different video. Our pre-trained network is generalizable to any videos; thus we avoid this main overhead compared to traditional optical flow based methods.

We summarize our contribution as follows:

a) An optical flow based video stabilization network: We proposed a novel neural network that takes the optical flow fields as the input and produces a pixel-wise warp field for each frame. Our neural network can be pre-trained and generalized to any videos. The details are discussed in Sec. 5.5.

b) Frequency domain regularized training: We propose a frequency domain loss function that enables learning with optical flow fields. We will show the necessity of this loss function in Sec. 5.5.1.

c) Robust video stabilization pipeline: We propose a pipeline that is robust to moving occlusion and optical flow inaccuracy by applying PCA Flow to video stabilization. The design is demonstrated in Sec. 5.3. In Sec. 5.7, we will show that our method generates better results compared to the state-of-the-art optimization based and deep learning based video stabilization methods. Our method also achieves $\sim$3x speed improvement compared to optimization based methods.

5.2 Related Works

In this section, we summarize the traditional physically based video stabilization methods and the recent deep learning based methods. Most existing video stabilization works are 2D physically based methods. The methods below all involve 2D feature tracking. The difference is mainly from the method for feature track smoothing and stable frame generation. Buehler et al.[BBM01] re-render the frames at smoothed camera positions using the non-metric IBR algorithm. Matsushita et al.[MOG+06] and Gleicher and Liu[GL08] use simple 2D full-frame transformations to warp the original frames. Liu et al.[LYTS13] uses a grid to warp the frames and smoothes the enclosing feature tracks. Grundmann et al.[GKE11] proposed an L1 optimal camera path for smoothing the feature tracks. Liu et al.[LGW+11] extracts and smoothes eigen-trajectories. Goldstein and Fattal[GF12] constrain the feature track smoothing with the epipolar geometry. Wang et al.[WLHL13] also keep the relative position of 2D feature points.

In addition to these 2D physically based methods, Liu et al.[LGJA09] first reconstruct the 3D position of the feature points and camera positions, then smooth the camera trajectory and reproject the feature points to new camera positions. Sun[Sun12] and Smith et al.[SZJA09] also use 3D information, but they require depth cameras and light field cameras respectively.

Later works use optical flow and smooth the motion at the pixel level. SteadyFlow[LYTS14] smoothes the motion vector changes on each pixel using iterative Jacobi-based optimization. Our method in Chapter 4 track the pixel motion using the optical flow. We optimized the neural network weights that generate the warp field, instead of solving for the warp field directly. However, this optimization must be repeated for each new video. Our method in this chapter also uses a neural network to infer the pixel-wise warp field, but the network is pre-trained and can be generalized to any videos. Moreover, as we discussed in Sec. 5.1, using optical flow in video stabilization leads to fundamental problems. Our method in this chapter is designed specifically to overcome these problems.

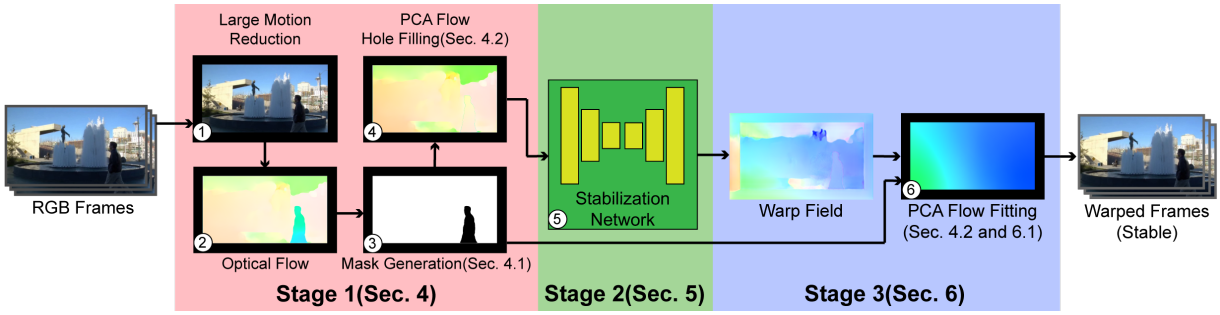


Figure 5.1: *The 3-stage pipeline of our algorithm. ①In the first stage(Sec. 5.4) we initially stabilize the video with translation and rotation. ②We compute the optical flow between consecutive frames. ③We generate a mask for each frame to indicate the valid regions for stabilization(Sec. 5.4.1). ④The invalid regions are inpainted using PCA Flow(Sec. 5.4.2). ⑤In the second stage(Sec. 5.5), our stabilization network infers the warp field from the inpainted optical flow fields. ⑥In the third stage(Sec. 5.6), we fit the PCA Flow to the raw warp field and use the smoothed warp field to warp the input video.*

Recent works start to apply deep learning to video stabilization. Xu et al.[XHW+18] uses the adversarial network to generate a target image to guide the frame warping. Wang et al.[WYL+19] uses a two branch Siamese network to generate a grid to warp the video frames. These networks take color frames as input and are trained with the DeepStab dataset, which contains stable and unstable video pairs. Deep learning methods enable near real-time performance in video stabilization. Visually, the results of these works are not as good as traditional methods. There are two potential reasons for the weak performance of deep learning in video stabilization. First, the video stabilization is a spatial transformation problem. The color images contain rich texture information, but the inter-frame spatial relation remains vague. Wang et al.[WYL+19] uses ResNet50 directly without any consideration of spatial transformation. Xu et al.[XHW+18] added spatial transformer modules to the adversarial network, but training a single network to infer spatial transformation of multiple frames only from color frames is difficult. Second, the dataset used in the training is not large enough. To our knowledge, the DeepStab dataset[XHW+18] is the only dataset for the learning of video stabilization and only contains 60 videos. For each video, the color frames are highly similar. Training an RGB based network with this dataset is essentially overfitting. Instead of trying to solve the video stabilization in

an end-to-end fashion, we separate the task into two parts. We first use FlowNet2[IMS+17] to compute the spatial correspondence between frames, then train a network to smooth the motion fields provided by FlowNet2. This makes the training easier and yields better results compared to networks trained end-to-end.

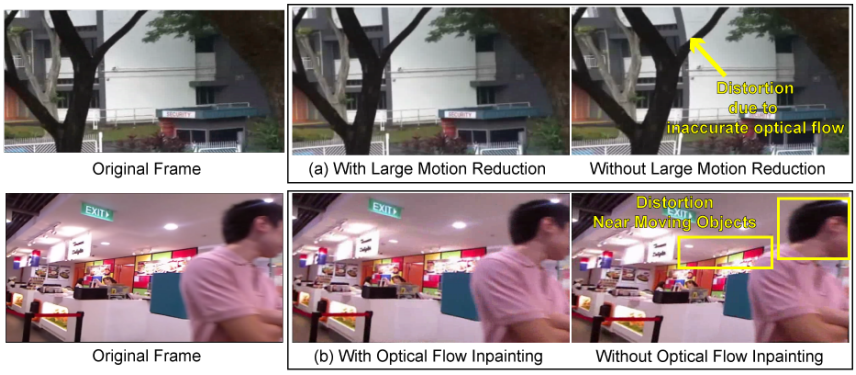


Figure 5.2: *The visual comparison of the results (a)with and without large motion reduction, (b)with and without masking and optical flow inpainting. The results contain distortion if large motion is not removed since the optical flow is not accurate. The distortion is also introduced by the moving object, if we do not use masks and inpaint the moving object regions.*

5.3 Pipeline

The pipeline of our algorithm is shown in Fig. 5.1. Stage 1 is the pre-processing. We remove the large motions in the video in the first step. We compute SURF features[BTVG06] and their matches between consecutive frames, then compute the affine transformations. The translation and rotation components of the affine transformations are smoothed by a simple moving average with a window size 40. The frames are transformed using the affine transformation to obtain the smoothed positions. The optical flow is computed with the state-of-the-art neural network FlowNet2[IMS+17] on the smoothed video sequence. The purpose of removing large motions is to increase the accuracy of the optical flow. In Fig. 5.2(a), we show a visual comparison of the final result versus only using the raw input. Large motion reduction helps avoid large displacement in the optical flow and warp fields, which usually introduce distortion in the results.

In the next step, based on a few criteria which will be discussed in Sec. 5.4.1, we generate a mask for each frame indicating the region where the optical flow is accurate. We inpaint the inaccurate regions using the first 5 principal components proposed in PCA Flow[WB15]. The coefficients are computed by fitting the principal components to the valid regions. In Fig. 5.2(b), we show an example using the raw optical flow without masking and inpainting. The person introduces significant distortion in the background due to the motion discontinuity. The analysis of the cause of this artifact and the details of motion inpainting will be discussed in Sec. 5.4.2.

The second stage is our stabilization network. The input of the network is the inpainted optical flow field. The network generates a per-pixel warp field for each frame, which compensates for the frame motion. In Sec. 5.5, we will discuss the loss function(Sec. 5.5.1) and the training process(Sec. 5.5.2).

The third stage is the post-processing. Since the optical flow in the invalid regions is inpainted, local discontinuities can be introduced at the valid/invalid boundaries. To ensure the continuity in the warp field, similar to stage 1, we fit the first 5 principal components to the warp fields in the valid regions. However, in stage 3, we replace the raw warp fields with the resulting low-frequency fits. We will discuss the necessity of this step in Sec. 5.6.1. Finally, we use the low-frequency warp fields to warp the input video. The warped video is cropped to a rectangle as the output.

5.4 Pre-Processing

In Sec. 5.3, we introduced the 3 stages of our pipeline: preprocessing(stage 1), stabilization network(stage 2) and warp field smoothing(stage 3). For stage 1, we discussed the large motion reduction and the optical flow computation in Sec. 5.3. In this section, we demonstrate the mask generation and the PCA Flow fitting in stage 1.

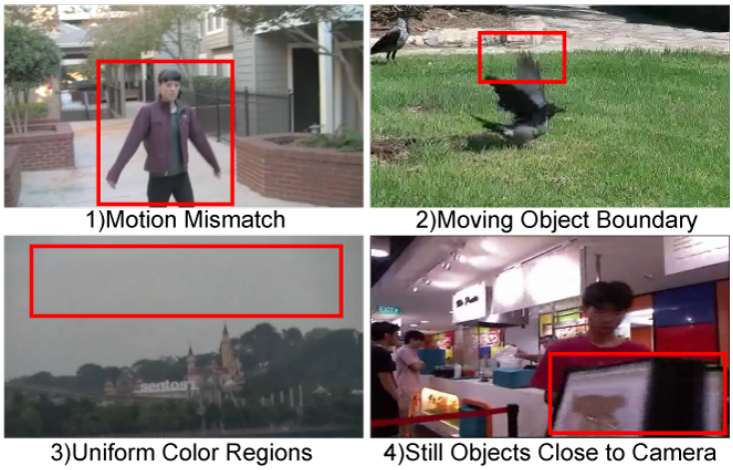


Figure 5.3: *Four types of scenarios in which optical flow can potentially be inaccurate or cause problems. Red boxes indicate example regions we refer to.*

5.4.1 Mask Generation

As we discussed in Sec. 5.1, using optical flow as the reference in video stabilization potentially suffers from reliability issues. We summarize these problems into four types which are shown in Fig. 5.3: **1)** Motions of moving objects do not match frame motion. **2)** Inaccurate at moving object boundaries. **3)** Inaccurate in uniform color regions due to the lack of motion information. **4)** Large motion of still objects due to parallax.

Our goal is to identify these regions, and generate a mask M so that $M = 0$ for these regions and $M = 1$ otherwise.

Denote the optical flow from frame I_n to frame I_{n+1} as $\mathbf{F}_n$. To detect type 1 regions, we use the pre-trained semantic segmentation network[ZZP+18, ZZP+17] to detect 11 kinds of possible dynamic object regions in I_n: person, car, boat, bus, truck, airplane, van, ship, motorbike, animal and bicycle. Note that these objects are not necessarily moving in the scene. Therefore, in these regions, we set $M_n(\mathbf{p}) = 1$ for any pixel $\mathbf{p}$ that satisfies $\|\mathbf{F}_n(\mathbf{p}) - \overline{\mathbf{F}_n}\|_2 < 5$, where $\overline{\mathbf{F}_n}$ is the mean motion of the entire frame.

In type 2 regions, the value of the optical flow changes significantly, causing a large local standard deviation. We compute the moving standard deviation with a 5×5 window, forming

the standard deviation map $\Delta\mathbf{F}_n$. We set $M_n(\mathbf{p}) = 0$ if $\Delta\mathbf{F}_n(\mathbf{p}) > 3\overline{\Delta\mathbf{F}_n}$ where $\overline{\Delta\mathbf{F}_n}$ is the mean standard deviation map value.

To detect type 3 regions, we compute the gradient image of frame I_n, denoted as ∇I_n. We set $M_n(\mathbf{p}) = 0$ if $\nabla I_n < 8$, since a smaller gradient value indicates less color variation.

For type 4 regions, we simply set $M_n(\mathbf{p}) = 0$ if $\|\mathbf{F}_n(\mathbf{p}) - \overline{\mathbf{F}_n}\|_2 > 50$, since the motion can only be very large in a large motion removed video if the object is very close to the camera.

We show sample masks generated using the metrics above in Fig. 5.4.

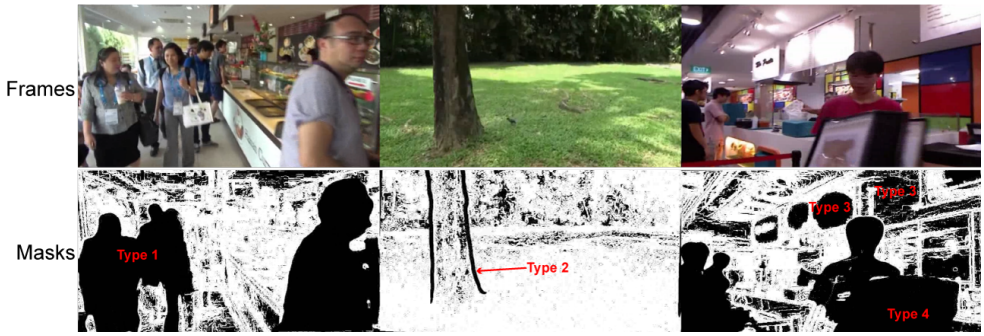


Figure 5.4: *Sample masks generated using our metrics described in Sec. 5.4.1. The four types of invalid regions are marked in the mask images.*

5.4.2 PCA Flow Fitting

To inpaint the motion vectors in the $M_n = 0$ regions, we fit the first 5 principal components proposed by PCAFlow[WB15] to the $M_n = 1$ regions. Since the first 5 principal components of PCAFlow are spatially smooth, we can expect the $M_n = 0$ regions are filled with reasonable values that obey the overall optical flow field. We reshape and stack the horizontal and vertical principal components into matrices $\mathbf{Q}_x$ and $\mathbf{Q}_y \in \mathbb{R}^{wh\times 5}$ respectively, where wh is the frame size. Similarly, we also reshape the optical flow field to $\mathbf{F}_{n,x}$ and $\mathbf{F}_{n,y} \in \mathbb{R}^{wh\times 1}$. For simplicity, we omit the subscript x and y. The fits below are computed independently for the horizontal and vertical directions. For each frame with mask M_n, we select the corresponding rows in $\mathbf{Q}$ and $\mathbf{F}$ where $M_n = 1$, forming the frame-specific principal components $\tilde{\mathbf{Q}}_n$ and valid optical flow matrix $\tilde{\mathbf{F}}_n$.

Finding the coefficients $\mathbf{c_n} \in \mathbb{R}^{5\times 1}$ to fit the valid optical flow $\tilde{\mathbf{F}}_n$ forms a traditional least squares problem:

$$\min_{\mathbf{c}_n} \left\| \tilde{\mathbf{Q}}_n \mathbf{c_n} - \tilde{\mathbf{F}}_n \right\|_2 + \eta \left\| \mathbf{c}_n \right\|_2 \tag{5.1}$$

where $\eta = 0.1$ is the regularization term. The solution of this problem is:

$$\mathbf{c}_n = (\tilde{\mathbf{Q}}_n^T \tilde{\mathbf{Q}}_n + \eta \mathbf{I})^{-1} \tilde{\mathbf{Q}}_n^T \tilde{\mathbf{F}}_n \tag{5.2}$$

We replace the optical flow values in $M_n = 0$ regions with the fitted PCA Flow $\mathbf{Q}_n \mathbf{c}_n$. The PCA Flow inpainted optical flow matrices for the horizontal and vertical directions are combined and reshaped back to the inpainted optical flow field $\widehat{\mathbf{F}}_n \in \mathbb{R}^{w\times h\times 2}$.

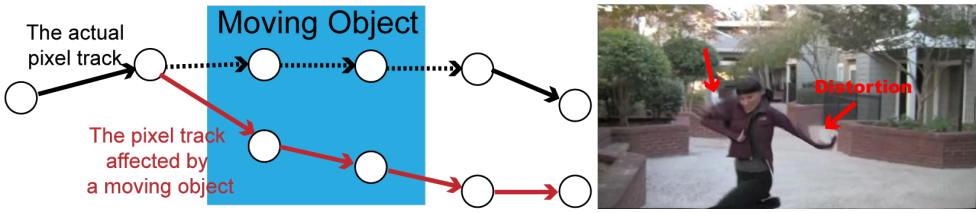


Figure 5.5: *The effect imposed by moving objects. The circles represent pixels and the arrows represent motion vectors evolving with time. The pixel can deviate from the actual track due to the moving object, resulting in a wrong warp field. The image on the right shows an example. The red arrows point out the distortion introduced by the moving object.*

5.4.3 Discussion

We demonstrate the necessity of using the mask in Fig. 5.5, in which we depict a 1D abstraction of the optical flow sequence. The moving object can cause a deviation in the motion vector that enters its region from the background, leading to a different pixel track from the actual motion pattern. Stabilizing the video in this scenario introduces distortion.

Applying the mask and stabilizing the valid regions alone still introduces distortion around moving objects. Figure 5.6 depicts an example of stabilizing only the valid regions. The mask M_n breaks the pixel track, making the pixels that connect to the masked pixels now only connect to

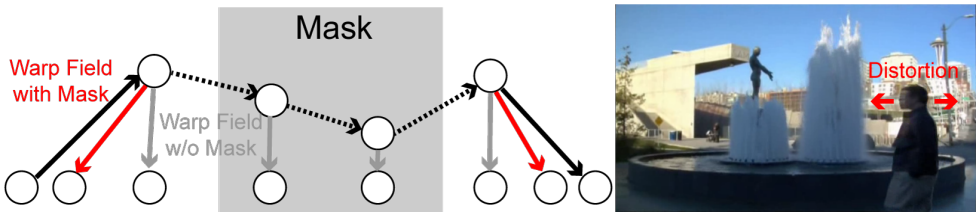


Figure 5.6: *Only stabilizing the valid regions will cause distortion in the warp field (red arrows) since the pixels are only constrained by the valid pixels connecting to it. The image on the right shows an example of this case.*

one correspondence. These pixels can move freely, causing distortion artifacts around the masked moving objects. Therefore, we need to inpaint the optical flow in the $M_n = 0$ regions so that the pixels connecting to these regions are constrained properly.

5.5 Network and Training

In this section, we introduce our video stabilization network. Our network follows the structure proposed by Zhou et al.[ZTF+18]. The network has a fixed number of input channels and can only take a segment of the optical flow sequence. Intuitively, the stabilization can handle low-frequency shake better if more frames are stabilized together since the network can access more global motion information. On the other hand, processing more frames together leads to a larger number of network weights and more difficulty in training. Taking all these factors into consideration, we use 20 frames of optical flow fields as the input of our network(representing the motion of a 21-frame video segment). Our network infers 19-frame warp fields for the video frames, excluding the first and the last frame. In the network structure of Zhou et al.[ZTF+18], we set the number of input channels of layer *conv1_1* to 20 and the number of output channels of layer *conv7_3* to 19. In Sec. 5.5.1, we define a loss function that enables the training of this network for our application. We will also introduce the training process of our network in Sec. 5.5.2. Note that to make our network be able to stabilize arbitrary long videos, we propose a sliding window schedule that will be discussed in Sec. 5.6.2.

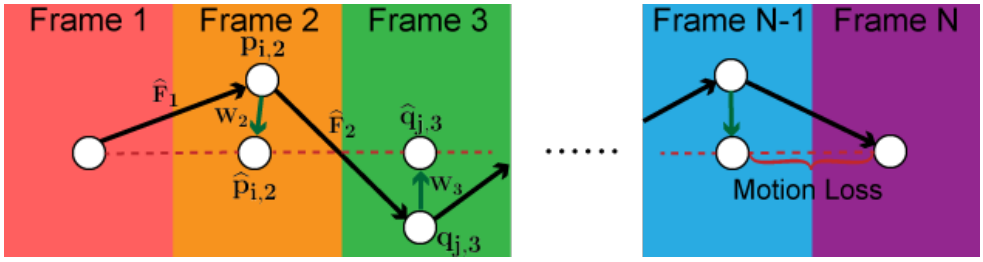


Figure 5.7: *A 1D abstraction of the motion loss. The loss indicates the average distance between corresponding pixels in each frame.*

5.5.1 Loss Functions

Denote a pixel at frame n as $\mathbf{p}_i$, where we unroll the pixel coordinates to index i. As discussed in Sec. 5.4.2, denote the PCA Flow inpainted optical flow from frame n to frame $n+1$ as $\widehat{\mathbf{F}}_n$. By definition, the correspondence of $\mathbf{p}_{i,n}$ in frame $n+1$, $\mathbf{q}_{j,n+1}$, can be represented as:

$$\mathbf{q}_{j,n+1} = \mathbf{p}_{i,n} + \widehat{\mathbf{F}}_n(\mathbf{p}_{i,n}) \tag{5.3}$$

Denote the output warp field for frame n as W_n. The warped position of a pixel $\mathbf{p}_{i,n}$ is defined as:

$$\widehat{\mathbf{p}}_{i,n} = \mathbf{p}_{i,n} + \mathbf{W}_n(\mathbf{p}_{i,n}) \tag{5.4}$$

Similarly, the warped position of its correspondence $\mathbf{q}_{j,n+1}$ an be written as:

$$\widehat{\mathbf{q}}_{j,n+1} = \mathbf{q}_{j,n+1} + \mathbf{W}_{n+1}(\mathbf{q}_{j,n+1}) \tag{5.5}$$

For a video segment with N frames, the number of optical flow fields between consecutive frames is $N-1$. Intuitively, our goal is to apply the warp field to every pixel so that the distance between correspondences are minimized. In Fig. 5.7, we depict a 1D abstraction of the motion loss. Note that we must fix the warp field to zero for the first and the last frame, i.e. $\mathbf{W}_1 = \mathbf{W}_N = 0$. In other words, the network only produces the warp field for the intermediate frames. Therefore,

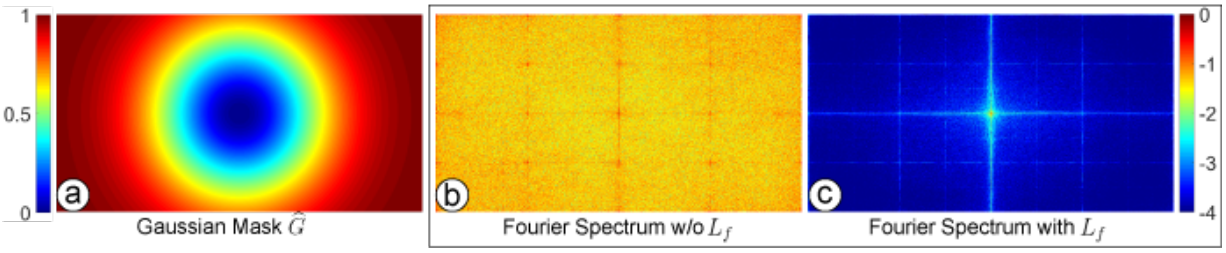


Figure 5.8: *The inverted Gaussian map*ⓐ *and an example spectrum of a warp field estimated with*ⓒ *and without*ⓑ *the frequency domain loss. The magnitude of the spectrum is shown in the* $\log_{10}$ *domain. The network can learn to produce a significantly smoother warp field with* L_f ⓒ.

we seek to find the shortest path to move the pixels in the first frame to their destination in the last frame instead of aligning all the frames. We define the motion loss as:

$$L_m = \sum_{n=1}^{N-1} \sum_{i} \left\| \widehat{\mathbf{p}}_{i,n} - \widehat{\mathbf{q}}_{j,n+1} \right\|_2 \tag{5.6}$$

In addition, we also seek to enforce the spatial smoothness of the warp fields. There are various kinds of constraints for enforcing spatial smoothness, e.g. total variation and the linear warp field constraint proposed in Chapter 4. However, the total variation constraint is strong in constraining local noise but weak in constraining distortions. The linear warping constraint is difficult to control since a strong constraint limits the warping flexibility to handle large scale non-linear motions, while a weak constraint will not constrain local distortions properly. In our method, we seek to unify the need for suppressing warp field noise and avoiding local distortion without affecting the flexibility of compensating global motions. Therefore, we propose to constrain the warp field in the frequency domain. Intuitively, the noise usually increases the high-frequency energy, while the local distortion increases the mid-frequency energy. The goal is to increase the low-frequency energy in the warp field, encouraging global warping and suppressing local warping and noise. This can be achieved by weighting the Fourier transform of the warp field in the training process. Computationally, we compute the 2D Fourier transform of each output warp field, then weight the spectrum by an inverted Gaussian map shown in Fig. 5.8. In our experiment, we generate the Gaussian map $\mathbf{G}$ with $\mu = 0$ and $\sigma = 3$, inverted by its maximum

value, and normalized by the maximum value:

$$\widehat{\mathbf{G}} = (\max(\mathbf{G}) - \mathbf{G}) / \max(\mathbf{G})$$

The frequency domain loss is defined as:

$$L_f = \sum_{n=2}^{N-1} \left\| \widehat{\mathbf{G}} \cdot \mathcal{F}\mathbf{W}_n \right\|_2 . \tag{5.7}$$

In this equation, the Fourier spectrum of the output warp field $\mathcal{F}\mathbf{W}_n$ is also normalized by its maximum value. Also note that the DC term of the inverted Gaussian map $\widehat{\mathbf{G}}$ is not used since we only encourage a low-frequency warp field but not a uniform warp field.

In the following section, we will discuss the usage of these loss functions in the training process.

5.5.2 Training

Dataset For the training of our network, we need a dataset with a large number of unstable videos. Existing video stabilization datasets, DeepStab[WYL+19](60 videos) and NUS[LYTS13](174 videos), do not contain enough motion pattern and color variation. In our training phase, we select the RealEstate10K[ZTF+18] dataset which contains stable videos with a large number of color variations. For each training sample, we randomly select 20 frames from a random video. To produce an unstable video, we simply perturb every frame other than the first frame and last frame using random 2D affine transformation. The parameters of this random 2D affine transformation are: scaling $U[0.9, 1.1]$, translation (percentage w.r.t the frame size) $U[-5\%, 5\%]$, rotation $U[-5^\circ, 5^\circ]$ and shear $U[-5^\circ, 5^\circ]$. The perturbed video forms the input of our network.

Training Phases We summarize our training process into two phases. In the first phase,

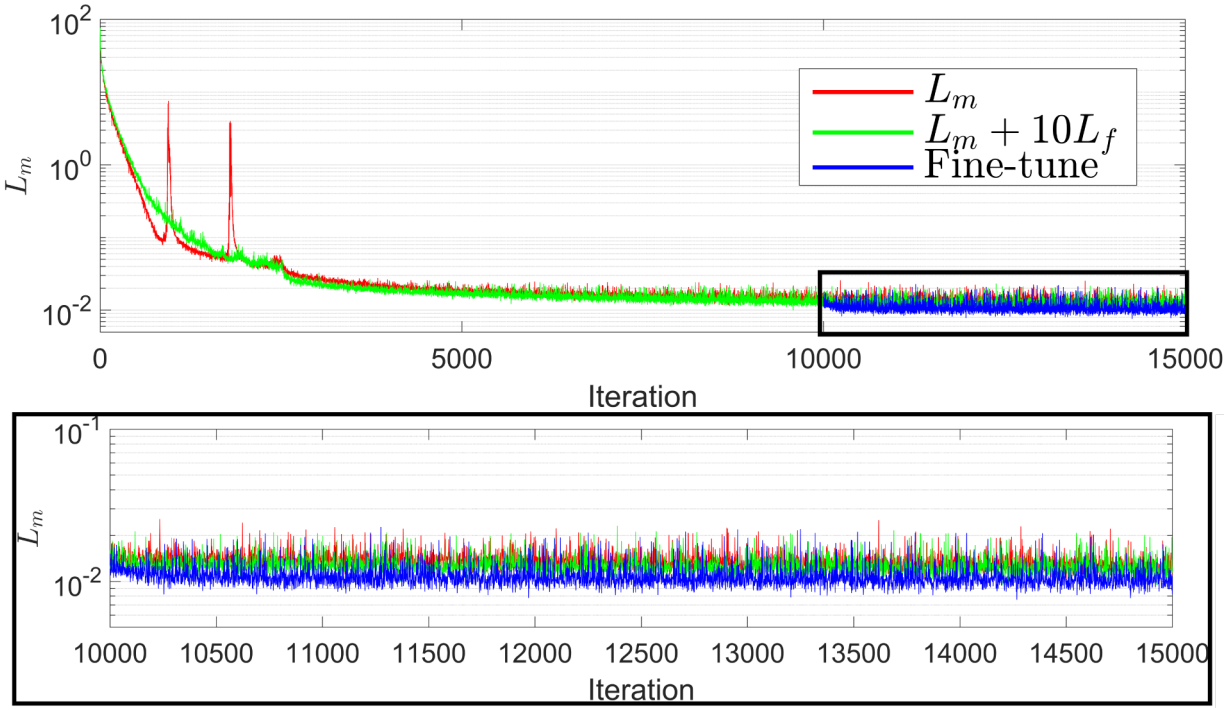


Figure 5.9: *The comparison of the motion loss L_m in different training schedules. The x-axis is the number of iterations and the y-axis is the L_m value. The figure below is a zoom-in version of the black box region of the upper figure. The red curve represents the training with L_m only. The green curve represents the training with $L_m + 10 * L_f$. The blue curve represents our training schedule. The frequency domain loss helps the first training phase so that the fine-tuning phase can achieve a lower motion loss.*

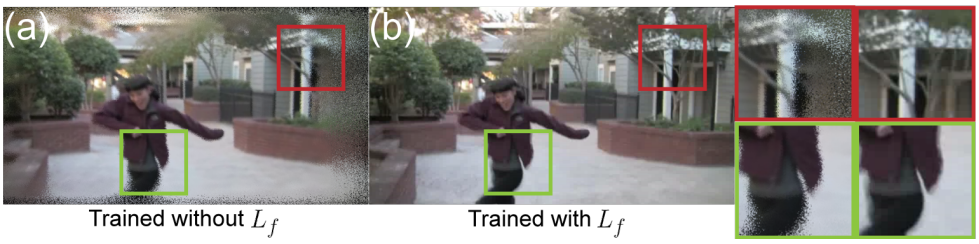


Figure 5.10: *The visual comparison of (a)the warped frames using the raw outputs of the networks trained with and (b)without L_f. The red and green boxes indicate the noisy regions. The frequency domain loss helps to improve the quality of the warp field.*

we set the loss function as:

$$L_1 = L_m + 10 * L_f \tag{5.8}$$

After training for 10000 iterations, we enter the second training phase, in which we switch the loss function to:

$$L_2 = L_m \tag{5.9}$$

and fine-tune the network for another 5000 iterations. We use the Adam optimizer[KB15] with $\beta_1 = 0.9$ and $\beta_2 = 0.99$. The learning rate is set to 10^{-4} for the first 2500 iterations and fixed to 10^{-5} for the rest of the training process.

To justify this training schedule, in Fig. 5.9, we plot the value of residual motion L_m which mainly indicates the training progress. The Case-I (red curve) represents the training with L_m only. The Case-II (green curve) represents the training with loss set to $L_m + 10 * L_f$. It can be observed that using L_f helps in making L_m descend to a lower value and expedite the training process. In Case-I, we observe that although the optical flow is spatially smooth, the output warp field usually contains high-frequency noise. The noise makes the network difficult to train in Case-I, especially in the early stages(spikes appear in the red curve). By introducing L_f to Case-I, we intend to suppress the high-frequency noise and reduce the local minima. After Case-II converges(iteration 10000), we switch back to Case-I to fine-tune the network. The blue curve in Fig. 5.9 shows that our schedule achieves the lowest loss level. Figure 5.10 shows an output frame comparison between training with L_m only and our training schedule. Using L_f makes the raw warp field smoother.

5.6 Testing and Implementation Details

In this section, we will discuss the details in the third stage and testing.

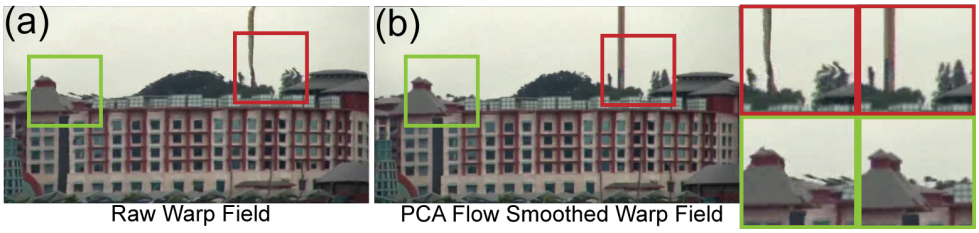


Figure 5.11: *The visual comparison of (a)the frames warped with the raw warp field and (b)the PCA Flow smoothed warp field. Due to the inpainting of the optical flow, the raw warp field may contain artifacts at the valid/invalid region boundaries.*

5.6.1 Warp Field Smoothing

Since the optical flow in the invalid regions is inpainted, our warp fields are only valid for the valid regions. The continuity of the warp field at valid/invalid boundaries is not guaranteed. Using the raw warp field introduces artifacts in the output, as shown in Fig. 5.11(a). Similar to the PCA Flow hole filling discussed in Sec. 5.4.2, we fit PCA Flow to the valid regions. In stage 3, we directly use the fitted PCA Flow as the warp field instead of the raw warp field. Figure 5.11(b) shows the result warped by the PCA Flow smoothed warp field.

5.6.2 Sliding Window

As discussed in Sec. 5.5, our network only takes 20 frames as the input. To handle a regular video, we use a sliding window approach for the testing phase.

The sliding window of our method works as shown in Fig. 5.12. In this figure, we use the notation $\mathbf{W}_{n,k}$ to represent the warp fields from different windows. The first index n is the frame number within a window, and the second index k is the window index. For each 20-frame window, the warp field for the first frame is already known from the previous window. We update the original optical flow as $\widehat{\mathbf{F}}_k - \mathbf{W}_{1,k-1}$ since the starting point of the motion vector is moved by $\mathbf{W}_{1,k-1}$ in the previous window. The updated optical flow is concatenated with the other optical flow fields as the input of the network, as shown on the left of window 2 and 3 in Fig. 5.12. We only use the first warp field produced by the network output to fit the principal component and

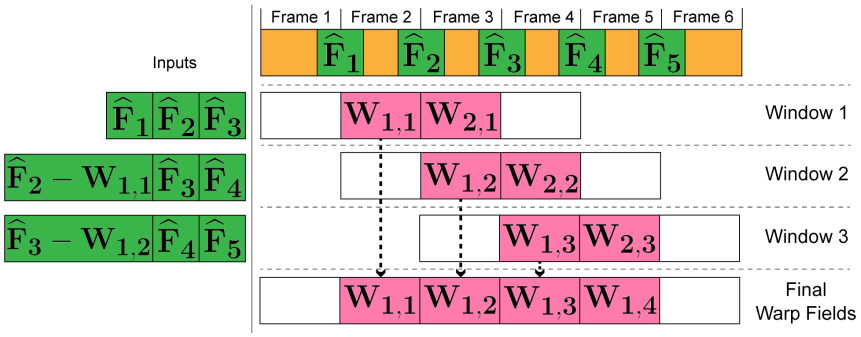


Figure 5.12: *The sliding window schedule for processing arbitrary long videos. For each window, we only accept the warp field for the second frame, e.g.* $\mathbf{W}_{1,1}$ *in window 1. In the next window(window 2), the inpainted optical flow from the first frame to the second frame(*$\widehat{\mathbf{F}}_2$*) is modified using the accepted warp field from the previous frame(*$\widehat{\mathbf{F}}_2 - \mathbf{W}_{1,1}$*).*

warp the second video frame. Then the window slides to the next frame and the process above repeats.

The disadvantage with the sliding window is that we are using 20 frames ahead of the current frame. However, the optical flow for these frames will be updated in the future windows, which should influence the current frame as well. For offline video stabilization, we can process the video with the sliding window for multiple passes. Between two passes, we re-compute the optical flow using the warped frames.

5.7 Results

In this section, we compare the results of our method with the state-of-the-art video stabilization methods. These methods are selected since they represent different approaches to the video stabilization problem. Grundmann et.al.[GKE11] uses the full-frame homography as the motion and warping model. Liu et.al.[LYTS13] uses a grid to analyze the local frame motion and warp the frames. Our method in Chapter 4 uses the dense optical flow as the motion model, and optimize CNN parameters that produce the pixel-wise warp field for each segment of a video. These methods belong to traditional optimization methods since they use traditional optimization and have to be re-run for a new video. We also compare with the most recent deep learning based

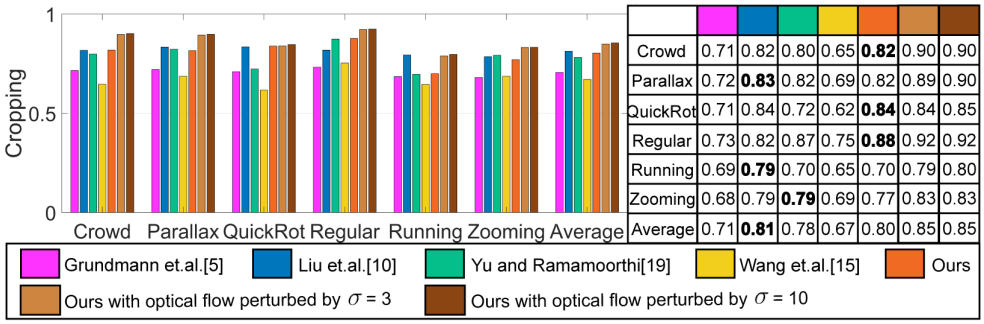
Figure 5.13: *The visual comparison of Grundmann et.al.[GKE11], Liu et.al.[LYTS13], Yu and Ramamoorthi[YR19a](Chapter 4), Wang et.al.[WYL+19] and our method. The artifacts are noted below the video stills and pointed out by arrows. To avoid introducing extra distortion, all the video stills are scaled while keeping the original aspect ratio.*

method, Wang et.al.[WYL+19], which uses a pre-trained network to directly infer a warp grid from colored input frames. We compare the results both visually and quantitatively.

Visual Comparison We provide visual comparisons in the supplementary video since most of the artifacts are only visible in the video. For the visual comparison of video stills, we selected a few difficult scenarios for the video stabilization task. Figure 5.13 shows the comparison of video stills from the comparison methods. Example 1 contains parallax effects with moving occlusions. Since Liu et.al.[LYTS13] uses a grid to warp the video, the region enclosed by a single cell is warped by the same homography. Therefore it generates a shear at the motion boundaries. It also introduces distortion in the uniform color regions(the body of the train), since estimating homography in these regions is difficult. Example 2 involves complex occlusions. Grid warping based methods, Liu et.al.[LYTS13] and Wang et.al.[WYL+19], produce local distortion due to motion mismatch. Our method in Chapter 4 introduces shear since the linear warping constraint enforces strong rigidity on the warp field, which tries to compensate for the motion. Our PCA Flow smoothed warp field provides more flexibility in warping compared to the grid used by Liu et.al.[LYTS13] and Wang et.al.[WYL+19], and the linear warping constrained warp field proposed in Chapter 4. Example 3 provides another example where Liu et.al.[LYTS13] produces shear at motion boundaries. Example 4 contains complex structures and Example 5 contains quick object motion. Both are challenging for optical flow based video stabilization methods. Our previous method in Chapter 4 produces significant distortion in these cases, since the linear warping constraint fails to constrain the local region in the warp field. Our PCA Flow based warp field avoids drastic compensation to optical flows and does not introduce artifacts in local regions.

The 2D full-frame homography method of Grundmann et.al.[GKE11] performs well on keeping original frame appearance, but in the supplementary video we will show that their temporal stability is inferior to that of comparison methods in the examples shown in Fig. 5.13. The pre-trained model proposed by Wang et.al.[WYL+19] failed to generate good results in most

of the videos, due to the difficulty in generalization.

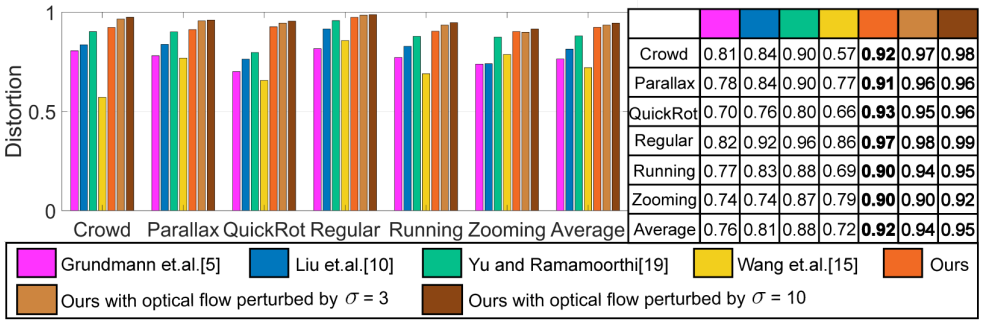


Crowd	0.71	0.82	0.80	0.65	**0.82**	0.90	0.90
Parallax	0.72	**0.83**	0.82	0.69	0.82	0.89	0.90
QuickRot	0.71	0.84	0.72	0.62	**0.84**	0.84	0.85
Regular	0.73	0.82	0.87	0.75	**0.88**	0.92	0.92
Running	0.69	**0.79**	0.70	0.65	0.70	0.79	0.80
Zooming	0.68	0.79	**0.79**	0.69	0.77	0.83	0.83
Average	0.71	**0.81**	0.78	0.67	0.80	0.85	0.85

Figure 5.14: *The cropping metric comparison of Grundmann et.al.[GKE11], Liu et.al.[LYTS13], Yu and Ramamoorthi[YR19a](Chapter 4), Wang et.al.[WYL+19] and our method. Each value is the result averaged by the category in the NUS dataset[LYTS13]. The last bar group is the average over all videos. The quantative values of the bars are shown in the table on the right. A larger value indicates a better result. The actual best result of each category before rounding is marked in bold font.*

Quantative Comparison For quantative comparison, we use the metrics proposed in Liu et.al.[LYTS13] to evaluate the quality of the results over the entire NUS dataset[LYTS13]. The values are averaged over each category. The cropping metric measures the frame size loss of the output video due to the warping and cropping. A larger value indicates a better frame size preservation. Figure 5.14 shows the cropping comparison. Our method maintains a large frame size similar to Liu et.al.[LYTS13] and Yu and Ramamoorthi[YR19a](Chapter 4), since the PCA Flow smoothed warp field does not introduce sharp warps that affect the cropping size. Our method is slightly worse than Liu et.al.[LYTS13] in the Running category since we have the large motion reduction step. The full-frame affine transformation removes large motions in the Running videos, but also leads to a smaller overlapping area and the final frame size. This can be easily avoided by using a smaller window size in the large motion reduction step.

The distortion metric measures the anisotropic scaling that leads to distortion in the result frames. A larger value indicates better preservation of the original shape of the objects in the video. Figure 5.15 shows the comparison of the distortion metric. Our per-pixel warp field introduces less distortion than the grid warping used in Liu et.al.[LYTS13] and Wang et.al.[WYL+19], and



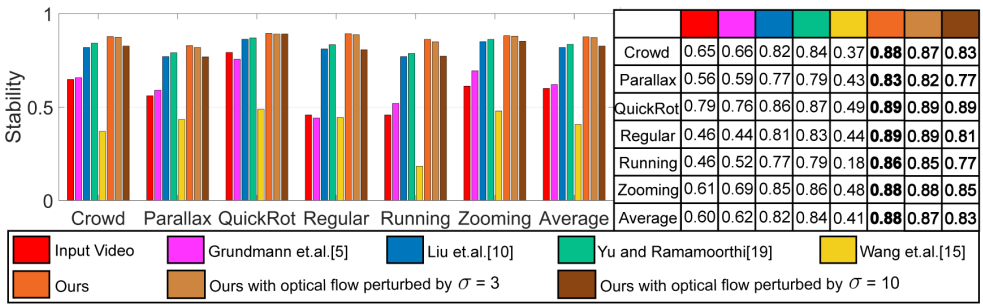
Crowd	0.81	0.84	0.90	0.57	**0.92**	0.97	0.98
Parallax	0.78	0.84	0.90	0.77	**0.91**	0.96	0.96
QuickRot	0.70	0.76	0.80	0.66	**0.93**	0.95	0.96
Regular	0.82	0.92	0.96	0.86	**0.97**	0.98	0.99
Running	0.77	0.83	0.88	0.69	**0.90**	0.94	0.95
Zooming	0.74	0.74	0.87	0.79	**0.90**	0.90	0.92
Average	0.76	0.81	0.88	0.72	**0.92**	0.94	0.95

Figure 5.15: *The distortion metric comparison of Grundmann et.al.[GKE11], Liu et.al.[LYTS13], Yu and Ramamoorthi[YR19a](Chapter 4), Wang et.al.[WYL+19] and our method. Each value is the result averaged by the category in the NUS dataset[LYTS13]. The last bar group is the average over all videos. The quantative values of the bars are shown in the table on the right. A larger value indicates a better result. The actual best result of each category before rounding is marked in bold font.*

the full-frame homography used in Grundmann et.al.[GKE11] in all the categories. Our method has less anisotropic scaling compared to the method in Chapter 4, since our warp field is more flexible than their linear warping constrained warp field. Therefore, our method performs better in preserving the original shape of the objects in the video. It can be also seen in the visual comparison that our PCA Flow smoothed warp field introduces less local distortion. Note that the comparison deep learning based method Wang et.al.[WYL+19] performs the worst in all the categories, implying the difficulty in generalization.

The stability metric measures the stability of the output video. A larger value indicates a more visually stable result. Our method achieves a better stability value compared to optimization based methods Grundmann et.al.[GKE11], Liu et.al.[LYTS13] and our method in Chapter 4. Our results are also significantly more robust than the deep learning based method Wang et.al.[WYL+19], since their results are even more unstable than the input video from the NUS dataset[LYTS13]. We will also show in the supplementary video that we achieve better stability values with less artifacts compared to these methods.

To evaluate the robustness of our method, we also conduct experiments with inaccurate



Crowd	0.65	0.66	0.82	0.84	0.37	**0.88**	**0.87**	**0.83**
Parallax	0.56	0.59	0.77	0.79	0.43	**0.83**	**0.82**	**0.77**
QuickRot	0.79	0.76	0.86	0.87	0.49	**0.89**	**0.89**	**0.89**
Regular	0.46	0.44	0.81	0.83	0.44	**0.89**	**0.89**	**0.81**
Running	0.46	0.52	0.77	0.79	0.18	**0.86**	**0.85**	**0.77**
Zooming	0.61	0.69	0.85	0.86	0.48	**0.88**	**0.88**	**0.85**
Average	0.60	0.62	0.82	0.84	0.41	**0.88**	**0.87**	**0.83**

Figure 5.16: *The stability metric comparison of Grundmann et.al.[GKE11], Liu et.al.[LYTS13], Yu and Ramamoorthi[YR19a](Chapter 4), Wang et.al.[WYL+19] and our method. Each value is the result averaged by the category in the NUS dataset[LYTS13]. The last bar group is the average over all videos. The quantative values of the bars are shown in the table on the right. A larger value indicates a better result. The actual best result of each category before rounding is marked in bold font.*

Table 5.1: Per-frame run time comparison

Grundmann et.al.[GKE11]	480ms
Liu et.al.[LYTS13]	1360ms
Yu and Ramamoorthi[YR19a](Chapter 4)	1610ms
Wang et.al.[WYL+19]	460ms
Ours	570ms

optical flow. We added Gaussian random noise with standard deviation $\sigma = 3$ and $\sigma = 10$ on the input optical flow to the network. In Fig. 5.14, Fig. 5.15 and Fig. 5.16, we observe that the inaccuracy in the optical flow leads to less cropping and distortion but worse stability. This indicates that the more inaccuracy in the optical flow, the more regions are identified as invalid regions in the stage 1 of our method. The network tends to warp the frame less since it receives less motion information. As shown in Fig. 5.16, our method is robust to inaccurate optical flow. Our method still maintains comparable level of stability even with optical flow perturbed by Gaussian noise with $\sigma = 10$.

Also note that since our method is a deep learning based method, the speed of our method is faster than the optimization based methods. Our network and pipeline are implemented with PyTorch and Python. Table. 5.1 is a summary of per-frame runtime for comparison methods. All the timing is performed on a desktop with an RTX2080Ti GPU and an i7-8700K CPU. On

average, our unoptimized method takes 270ms in stage 1 and 300ms in stage 2 and 3. Our method achieves better visual results and somewhat better quantitative results compared to optimization based methods Liu et.al[LYTS13] and our method in Chapter 4 but gives $\sim$3x speed up. Our method has only a slight computation time loss compared to the simple 2D method of Grundmann et.al.[GKE11] and deep learning based method of Wang et.al.[WYL$^+$19], but generates significantly better visual and quantative results.

5.8 Summary

In this chapter, we proposed a novel deep learning based video stabilization method that infers the pixel-wise warp field for stabilizing video frames from the optical flow between consecutive frames. We also proposed a pipeline that detects invalid regions in the optical flow field, inpaints the invalid regions and smoothes the output warp field. The results show that our method is more robust than existing deep learning based methods and achieves visually and quantitatively better results compared to the state-of-the-art optimization based methods with a $\sim$3x speed improvement. Future works would be an end-to-end network that directly converts input videos to stabilized videos, and a dataset that enables the training of such a network.

This chapter is a reformatted version of the material as it appears in "Learning Video Stabilization Using Optical Flow," Jiyang Yu and Ravi Ramamoorthi in the IEEE Conference on Computer Vision and Pattern Recognition(CVPR), 2020 [YR20].The dissertation author was the primary investigator and author of this paper.

Chapter 6

Conclusion and Future Work

In this dissertation, we focus on working towards robust video stabilization methods that are practical for real-life uses. In Chapter 2, we proposed an optimization based selfie video method. We demonstrate that a 3D face model can be used to track the foreground motion of a selfie video, while optical flow can be used to track the background motion. By optimizing the foreground and background motion jointly, we generate stable selfie video while minimizing the cropping of output video. We also demonstrate how the ideas in the selfie video stabilization are posed as a learning based pipeline in Chapter 3. We designed a two branch 1D linear convolutional neural network that directly infers MLS warping from input background feature points and foreground 3D face model vertices. To train our network, we also proposed a selfie video dataset that contains 1005 videos, which is the first large scale selfie video dataset for deep learning. We also propose a grid approximation to the MLS warping and sliding window scheme that enables our method to run in an online fashion at a real-time rate.

In Chapter 4 and Chapter 5, we focus on general video stabilization. For general videos, there is no prior information of occluding objects as selfie videos. Our works follow two different ideas to overcome the occlusion and other complex effects in the video. First, we can directly impose a constraint on the output warp field without trying to remove these effects in the input. In

Chapter 4, we optimize a convolutional neural network for each individual input video to generate a per-pixel dense warp field that stabilizes the input. In the objective function, we constrain the output warp field to be similar to a linear warp field. Second, we can pre-process the input video to exclude occlusion and other effects before stabilization. In Chapter 5, we exploit a recent deep learning method to segment possible occluding objects and detect regions where the optical flow is inaccurate. We use PCAFlow to fit to valid optical flow regions and inpaint the invalid optical flow regions. After this pre-processing, our pre-trained stabilization network can blindly compensate the motion. Both these works produce superior results compared to the state-of-the-art optimization based and learning based video stabilization algorithms.

Our works in this dissertation provide an in-depth discussion of the challenges and possible solutions of robust video stabilization. We also demonstrate the possibility of using deep learning in video stabilization, which leads to the following future works. The first line of future work is a complete end-to-end deep neural network that directly generates stabilized frames from input frames, which is a way to break the qualitative limitation imposed by inaccurate feature detection and optical flow calculation. The second line of future work is to integrate sensor information in mobile platforms, e.g. inertial sensor, depth sensor, and gyroscope into the learning of video stabilization. These sensors enable more accurate and physics-based supervision compared to the hand-crafted loss in the image domain. The third line of future work is to extend the video stabilization into 2.5D space. As we discussed in this dissertation, 3D video stabilization methods require structure from motion or a depth camera, which are either prone to error in real-life cases or require specific hardware. On the other hand, recent works in 2.5D representations like multi-plane images, demonstrate promising results in new view synthesis. This also provides a new oppotunity for 3D video stabilization, where the motion can be interpreted as 3D camera motion and the stabilized frame rendering can utilize the more robust 2.5D multi-plane image representation.

Bibliography

[AEHKL+16] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Apostol (Paul) Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. In *arXiv:1609.08675*, 2016.

[BAAR14] Jiamin Bai, Aseem Agarwala, Maneesh Agrawala, and Ravi Ramamoorthi. User-assisted video stabilization. In *EGSR*, 2014.

[BBM01] Christopher Buehler, Michael Bosse, and Leonard McMillan. Non-metric image-based rendering for video stabilization. In *IEEE CVPR*, 2001.

[BBPV03] V. Blanz, C. Basso, T. Poggio, and T. Vetter. Reanimating Faces in Images and Video. *Computer Graphics Forum*, 22(3), 2003.

[BT17] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2D & 3D face alignment problem? (and a dataset of 230,000 3D facial landmarks). In *IEEE ICCV*, 2017.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *ECCV*, 2006.

[BV99] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3D faces. In *ACM SIGGRAPH*, 1999.

[BZS+07] Pravin Bhat, C. Lawrence Zitnick, Noah Snavely, Aseem Agarwala, Maneesh Agrawala, Michael Cohen, Brian Curless, and Sing Bing Kang. Using photographs to enhance videos of a static scene. In *EGSR*, 2007.

[CBZB15] Chen Cao, Derek Bradley, Kun Zhou, and Thabo Beeler. Real-time high-fidelity facial performance capture. *ACM Trans. Graph.*, 34(4), July 2015.

[CHZ14] Chen Cao, Qiming Hou, and Kun Zhou. Displaced dynamic expression regression for real-time facial tracking and animation. *ACM Trans. Graph.*, 33(4), July 2014.

[CK20] Jinsoo Choi and In So Kweon. Deep iterative frame interpolation for full-frame video stabilization. *ACM Trans. Graph.*, 39(1), January 2020.

[CKR+19] P. Chao, C. Kao, Y. Ruan, C. Huang, and Y. Lin. Hardnet: A low memory traffic network. In *IEEE International Conference on Computer Vision (ICCV)*, 2019.

[CT19] Bao Xin Chen and John K Tsotsos. Fast visual object tracking with rotated bounding boxes. 2019.

[CWZ+14] C. Cao, Y. Weng, S. Zhou, Y. Tong, and K. Zhou. Facewarehouse: A 3D facial expression database for visual computing. *IEEE Trans. Visual. and Comput. Graph.*, 20(3), March 2014.

[DLHT14] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Learning a deep convolutional network for image super-resolution. In *ECCV*, 2014.

[EKD+17] Gabriel Eilertsen, Joel Kronander, Gyorgy Denes, Rafal Mantiuk, and Jonas Unger. HDR image reconstruction from a single exposure using deep CNNs. *ACM Trans. Graph.*, 2017.

[FSGF16] Ohad Fried, Eli Shechtman, Dan B Goldman, and Adam Finkelstein. Perspective-aware manipulation of portrait photos. 2016.

[GF12] Amit Goldstein and Raanan Fattal. Video stabilization using epipolar geometry. *ACM Trans. Graph.*, 31(5), September 2012.

[GKE11] Matthias Grundmann, Vivek Kwatra, and Irfan Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *IEEE CVPR*, 2011.

[GL08] Michael L. Gleicher and Feng Liu. Re-cinematography: Improving the camerawork of casual video. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(1), October 2008.

[GZC+16] Pablo Garrido, Michael Zollhöfer, Dan Casas, Levi Valgaerts, Kiran Varanasi, Patrick Pérez, and Christian Theobalt. Reconstruction of personalized 3D face rigs from monocular video. *ACM Trans. Graph.*, 35(3), May 2016.

[HLSH18] Mingming He, Jing Liao, Pedro Sander, and Hugues Hoppe. HDR image reconstruction from a single exposure using deep CNNs. *ACM Trans. Graph.*, 2018.

[IBP15] Alexandru Eugen Ichim, Sofien Bouaziz, and Mark Pauly. Dynamic 3D avatar creation from hand-held video input. *ACM Trans. Graph.*, 34(4), July 2015.

[IMS+17] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE CVPR*, 2017.

[JSJ+18] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. Super slomo: High quality estimation of multiple intermediate frames for video interpolation. In *CVPR*, 2018.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Kin09] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[KJBL11] Alexandre Karpenko, David Jacobs, Jongmin Baek, and Marc Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. In *Stanford CS Tech Report*, 2011.

[Kop16] Johannes Kopf. 360° video stabilization. *ACM Trans. Graph.*, 2016.

[KTDG16] Till Kroeger, Radu Timofte, Dengxin Dai, and Luc Van Gool. Fast optical flow using dense inverse search. In *ECCV*, 2016.

[LCSP19] Yiming Lin, Shiyang Cheng, Jie Shen, and Maja Pantic. Mobiface: A novel dataset for mobile face tracking in the wild. In *The IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, 2019.

[Lef16] Stamatios Lefkimmiatis. Non-local color image denoising with convolutional neural networks. In *CVPR*, 2016.

[LGJA09] Feng Liu, Michael Gleicher, Hailin Jin, and Aseem Agarwala. Content-preserving warps for 3D video stabilization. *ACM Trans. Graph.*, 28(3), July 2009.

[LGW+11] Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. Subspace video stabilization. *ACM Trans. Graph.*, 30(1), February 2011.

[LHAY17] Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution. In *CVPR*, 2017.

[Liu09] Ce Liu. *Beyond pixels: exploring new representations and applications for motion analysis*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 2009.

[LJL+17] Kaimo Lin, Nianjuan Jiang, Shuaicheng Liu, Loong-Fah Cheong, Minh Do, and Jiangbo Lu. Direct photometric alignment by mesh deformation. 2017.

[LLZZ17] S. Liu, M. Li, S. Zhu, and B. Zeng. Codingflow: Enable video coding for video stabilization. *IEEE Transactions on Image Processing*, 26(7):3291–3302, 2017.

[LSD15] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[LTH+17] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, 2017.

[LTY+16] Shuaicheng Liu, Ping Tan, Lu Yuan, Jian Sun, and Bing Zeng. Meshflow: Minimum latency online video stabilization. In *European Conference on Computer Vision (ECCV)*, 2016.

[LYTS13] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Bundled camera paths for video stabilization. *ACM Trans. Graph.*, 32(4), July 2013.

[LYTS14] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Steadyflow: Spatially smooth optical flow for video stabilization. In *IEEE CVPR*, 2014.

[MOG+06] Yasuyuki Matsushita, Eyal Ofek, Weina Ge, Xiaoou Tang, and Heung-Yeung Shum. Full-frame video stabilization with motion inpainting. *IEEE Trans. Pattern Anal. Mach. Intell.(PAMI)*, 28(7), July 2006.

[NSR18] V. Nekrasov, Chunhua Shen, and I. Reid. Light-weight refinenet for real-time semantic segmentation. In *The British Machine Vision Conference (BMVC)*, 2018.

[SMW06] Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3), July 2006.

[SP04] Robert W. Sumner and Jovan Popović. Deformation transfer for triangle meshes. In *ACM SIGGRAPH*, 2004.

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE CVPR*, 1994.

[STWL19] Fuhao Shi, Sung-Fang Tsai, Youyou Wang, and Chia-Kai Liang. Steadiface: Real-time face-centric stabilization on mobile phones. In *IEEE International Conference on Image Processing (ICIP)*, 2019.

[Sun12] Jian Sun. Video stabilization with a depth camera. In *IEEE CVPR*, 2012.

[SWTC14] Fuhao Shi, Hsiang-Tao Wu, Xin Tong, and Jinxiang Chai. Automatic acquisition of high-fidelity facial performances using monocular videos. 2014.

[SZJA09] Brandon M. Smith, Li Zhang, Hailin Jin, and Aseem Agarwala. Light field video stabilization. In *IEEE ICCV*, 2009.

[TYL17] Ying Tai, Jian Yang, and Xiaoming Liu. Image super-resolution via deep recursive residual network. In *CVPR*, 2017.

[TZS+16] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *IEEE CVPR*, 2016.

[UVL18] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. *CVPR*, 2018.

[VJ01] Paul A. Viola and Michael J Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE CVPR*, 2001.

[WB15] Jonas Wulff and Michael J. Black. Efficient sparse-to-dense optical flow estimation using a learned basis and layers. In *IEEE CVPR*, 2015.

[WLHL13] Yu-Shuen Wang, Feng Liu, Pu-Sheng Hsu, and Tong-Yee Lee. Spatially and temporally optimized video stabilization. *IEEE Trans. Visual. and Comput. Graph.*, 19(8), Aug 2013.

[WYL+19] Miao Wang, Guo-Ye Yang, Jin-Kun Lin, Song-Hai Zhang, Ariel Shamir, Shao-Ping Lu, and Shi-Min Hu. Deep online video stabilization with multi-grid warping transformation learning. *IEEE Transactions on Image Processing*, 2019.

[XHW+18] Sen-Zhe Xu, Jun Hu, Miao Wang, Tai-Jiang Mu, and Shi–Min Hu. Deep video stabilization using adversarial networks. *Computer Graphics Forum*, 2018.

[YR19a] Jiyang Yu and Ravi Ramamoorthi. Robust video stabilization by optimization in cnn weight space. In *IEEE CVPR*, 2019.

[YR19b] Jiyang Yu and Ravi Ramamoorthi. Selfie video stabilization. *IEEE Trans. Pattern Anal. Mach. Intell.(PAMI)*, July 2019.

[YR20] Jiyang Yu and Ravi Ramamoorthi. Learning video stabilization using optical flow. In *IEEE CVPR*, 2020.

[YRC+20] Jiyang Yu, Ravi Ramamoorthi, Keli Cheng, Michel Sarkis, and Ning Bi. Real-time selfie video stabilization. In *arXiv:2009.02007*, 2020.

[ZLLL19] Xiangyu Zhu, Xiaoming Liu, Zhen Lei, and Stan Z Li. Face alignment in full pose range: A 3d total solution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(1), 2019.

[ZTF+18] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *ACM Trans. Graph.*, 2018.

[ZZP+17] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *IEEE CVPR*, 2017.

[ZZP+18] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal on Computer Vision*, 2018.