Real-Time Selfie Video Stabilization

Jiyang Yu[1,3] Ravi Ramamoorthi[1] Keli Cheng[2] Michel Sarkis[2] Ning Bi[2]
[1]University of California, San Diego [2]Qualcomm Technologies Inc. [3]JD AI Research, Mountain View
jiy173@eng.ucsd.edu ravir@cs.ucsd.edu {kelic,msarkis,nbi}@qti.qualcomm.com

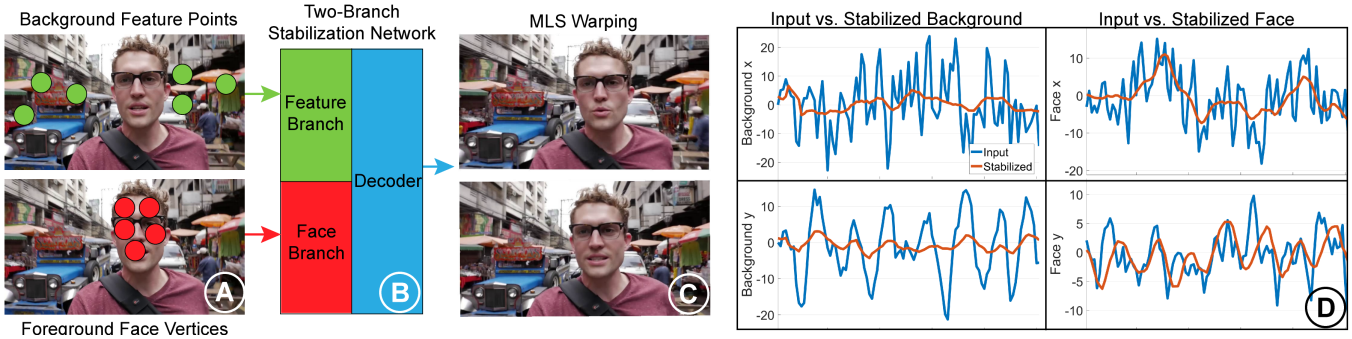

Figure 1. *Our method stabilizes selfie videos using Ⓐ background feature points and foreground face vertices in each frame. Ⓑ The two-branch stabilization network infers Ⓒ the moving least squares (MLS) warping for each frame. Ⓓ We show the face and background motion of the input vs. our stabilized 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.*

Abstract

We propose a novel real-time selfie video stabilization method. Our method 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 collect a selfie video dataset with 1005 videos, which is significantly larger than previous selfie video datasets. We also propose a grid approximation to the rigid moving least squares that enables the real-time frame warping. Our method is fully automatic and produces visually and quantitatively better results than previous real-time general video stabilization methods. Compared to previous offline selfie video methods, our approach produces comparable quality with a speed improvement of orders of magnitude. Our code and selfie video dataset is available at https://github.com/jiy173/selfievideostabilization.

1. Introduction

Selfie videos are pervasive in daily communications. However, capturing high quality selfie video is challenging without specialized stabilization devices like gimbals, which is not convenient, and may not even be allowed in some cases. On the other hand, from the perspective of algorithms, selfie video stabilization is also challenging. In general, there are three major steps in video stabilization algorithms. The first step is to detect the motion in the input video. Selfie videos have a significant foreground occlusion imposed by human, which is a common limitation of video stabilization algorithms since tracking the frame motion is difficult in the presence of large occlusion. The second step is to replan/stabilize the motion. In selfie videos, the motions in foreground/background are usually very different. Existing selfie video stabilization methods like [19] aim to stabilize the face. However, stabilizing according to only foreground results in significant shake in the background, and vice versa. The third step is the warping of the frames. For selfie videos, the users are sensitive to distortion on the human face. This requires high rigidity in the foreground warping while maintaining the flexibility in the background warping.

Critically, 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 [10], optical flow [15, 25, 4] and future motion information [6, 11]. A previous selfie video stabilization method [24] is an optimization based method and cannot achieve real-time performance. Although another selfie video stabilization work Steadiface [19] 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 paper, 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 shown in Fig. 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. 1Ⓐ). This makes our method significantly faster than the offline selfie video stabilization [24]. 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 [26] to fit a 3D mesh to video frames.

To warp the original frames into stabilized frames, we use the rigid moving least squares (MLS) [18] (Fig. 1Ⓒ). In our method, we directly use the background feature points as the warp nodes so that the face shape remains undistorted. Since the original MLS warping is computationally expensive, we use a grid approximation to maintain the real-time performance. Although faster warping methods exist, e.g. as-similar-as-possible warping in Liu et al. [14], MLS warping is necessary for our method. First, traditional grid warping requires an additional hyperparameter to regularize the grid shape. These terms usually conflict with the motion loss and manually setting the weight between visual distortion and stability is tricky. On the other hand, MLS warping guarantees rigidity implicitly and does not require human intervention. It also preserves the original shape of regions that lack warp nodes. Second and more importantly, our method is learning based instead of optimization based. In the traditional optimization process, it is easy to define the mapping between grid vertices and their enclosed feature points in the Jacobian. However, learning this spatial relation between feature points and grid vertices is difficult and suffers from generalization problems. In Sec. 6 and the supplementary video, we will show that our setup with MLS warping directly defined on unstructured warp nodes (feature points) is more effective than directly learning the grid like Wang et al. [21].

The core of our method is the stabilization network (Fig. 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. In addition, we find that removing activation layers used in traditional neural networks yields better results(supplementary Table d). The reason is that our formulation requires a linear 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, we will show in the supplementary material that direct optimization for this linear relationship is prohibitive in terms of computational efficiency and accuracy (supplementary Table c)[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 [25] and other domains [8] in previous works.

The contribution of our paper includes:

1) A novel selfie video stabilization network that enables real-time selfie video stabilization. Our network directly infers the moving least squares warp from the 2D feature points, stabilizing both the foreground face and background feature motion (Sec. 3.1 and Sec. 3.2). In Sec. 4.3 we will show that the structure of our network allows an optional control of stabilization focus.

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. 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. 4.1. The dataset enables the training of the foreground detection network and the stabilization network in our paper. We will make our dataset publicly available for face and video related researches.

[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.

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 [6], bundled camera paths [14], subspace video stabilization [11], video stabilization using epipolar geometry [5], content-preserving warps [10] and spatially and temporally optimized video stabilization [22]. These works all involve the detection of feature tracks and smoothing under certain constraints. Some works use optical flow [15, 25] or video coding [12] 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 [14] only smooth tracks formed by feature points. Falsely detected features can easily affect the optimization, especially when the window size is small. Moreover, [14] requires global motion information to achieve the reported result. One can expect performance to decrease if a short sliding window is applied. In Sec. 6 we will show that [14] already generates inferior results than ours using the entire video (Fig. 8 and Fig. 9). As we will discuss in Sec. 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 [14]. 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 [23] and estimate a warp grid from input frames [21]. These methods are difficult to generalize to videos in the wild. Other learning based works (e.g., [4]) 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. An existing selfie video stabilization method [24] uses the face centroid to represent the fore-

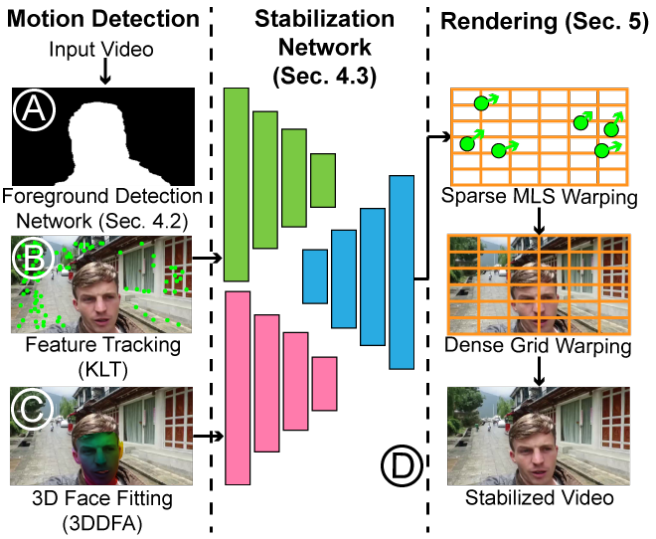

Figure 2. *The pipeline of our method. Ⓐ We first detect the foreground regions of the input video frame. Ⓑ The background motion is tracked using feature points. Ⓒ The foreground motion is tracked using 3D face vertices. Ⓓ We train a stabilization network to infer the displacement of the MLS warp nodes. Finally, we use a grid to approximate the MLS warping and generate the stabilized frame.*

ground motions while stabilizing the background motions. However, their method uses the optical flow to detect the background motion and the foreground mask, which is computationally expensive for real-time applications. Their method is also based on global motion optimization, which makes it impractical in online video stabilization. Our method does not require the dense optical flow computation and does not require future motion information, therefore is more efficient than their method.

Steadiface [19] 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 provides flexibility to handle non-linear distortions. Our method also models the face motion more accurately using a face mesh instead of face landmarks in [19]. Due to these limitations, Steadiface [19] 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 [19] in Fig 10(b) and the supplementary video.

MeshFlow [13] 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. 6, we will compare our result with selfie video stabilization [24], Steadiface [19], MeshFlow [13] and the state-of-the-art learning based approaches [4, 21]. We also compare with the bundled camera path video stabilization [14] representing a typical offline general video stabilization method as the reference.

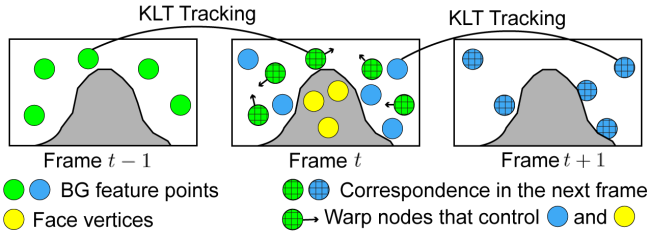

Figure 3. *The warping strategy of our method. The background feature points in the same color are in correspondence. The feature points with grid patterns are the warp nodes. The arrows represent the MLS warping operation. During the stabilization, both the feature points $\mathbf{P}_t$(solid blue points) and the face vertices $\mathbf{F}_t$(solid yellow points) are warped by the warp nodes $\mathbf{Q}_t$(grid green points).*

3. Overview of algorithm pipeline

Our pipeline is shown in Fig. 2. The pipeline consists of three major 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. For completeness, we summarize the notations used in our paper and supplementary material in supplementary Table a. The training of the neural networks mentioned below will be discussed in Sec. 4.

3.1. Motion Detection

As discussed in Sec. 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 foreground region of frame t. We show a sample foreground mask in Fig. 2Ⓐ. The details regarding the foreground detection network will be discussed in Sec. 4.2. For the background region where $\mathbf{M}_t = 0$, we use the Shi-Tomasi corner detector[20] to detect feature points in a frame and the KLT tracker to find their correspondences in the next frame, as shown in Fig. 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. 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 [26]. An example of a fitted 3D face mesh is shown in Fig. 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 paper, we denote the selected face vertices as $\mathbf{F}_t \in \mathbb{R}^{2\times 512}$, where t represents the frame index.

3.2. Stabilization

To stabilize the video, we use the rigid moving least square(MLS) warping[18] to warp the frames. In Fig. 3, we depict the warping strategy of a video sequence. The moving least square warping requires a set of warp nodes for

Figure 4. Ⓐ *Our selfie video dataset. From left to right: color frame, ground truth foreground mask, background feature points, 3D face mesh.* Ⓑ *Examples of the foreground mask detected with our trained foreground detection network.*

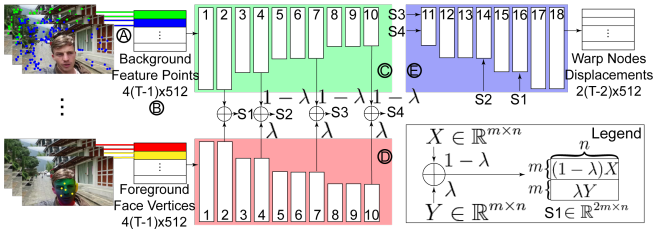

Figure 5. *Our stabilization network structure. On the left we show a sequence of input frames.* Ⓐ *The feature points and their correspondences in the next frame are concatenated as a* 4×512 *tensor.* Ⓑ *The tensors in the same window are concatenated to a large* $4(T-1) \times 512$ *tensor. The same 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 provided in the supplementary material Table b*

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 gridded green dots in Fig. 3). Besides all the pixels in frame t, the feature points $\mathbf{P}_t$ (solid blue dots) and the face vertices $\mathbf{F}_t$ (solid yellow 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) 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. We included the details of the MLS warping in supplementary material Algorithm 1. In this paper, we propose a convolutional network (Fig. 2Ⓓ) to infer the displacements of warp nodes $\widehat{\mathbf{Q}}_t - \mathbf{Q}_t$. In Sec. 4.3, we will discuss the training of this stabilization network.

3.3. Warping

More feature points(warp nodes) leads to less warping artifacts but longer time to detect and track. In our paper, we use 512 feature points as warp nodes in each frame, which is a tradeoff between visual quality and runtime performance. Details will be discussed in supplementary material Sec. C.2. 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. 5, we will demonstrate the details of the grid warping approximation.

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. 4.1), then discuss details of the foreground detection network (Sec. 4.2) and stabilization network (Sec. 4.3). In the supplementary meterial, we introduce a sliding window scheme to apply our stabilization network to arbitrarily long videos (Sec. A.2).

4.1. Dataset

Although large scale video datasets like Youtube-8M [1] 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 existing selfie video datasets proposed in [24](33 videos) and [9](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 [7] 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 [3] 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 [26]. In Fig. 4Ⓐ, we show a video still, the corresponding foreground mask, the background feature points and the 3D face mesh from our dataset. Our dataset will be made publicly available upon publication.

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 [16] for this segmentation task. Although there are more advanced structure for segmentation [17, 2], 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 in Sec. 3.2. The training uses Adam optimizer with a 10^{-3} learning rate and a binary cross entropy loss. Figure 4Ⓑ provides examples of the inferred masks on video frames outside our dataset. Note that the inferred mask is not perfect, but it is accurate enough to distinguish the foreground and the background.

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. 3.1. 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

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

where $W(\mathbf{P}_t; \mathbf{Q}_t, \widehat{\mathbf{Q}}_t)$ is the MLS warping function as mentioned in Sec. 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 supplementary material 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:

$$L_f = \sum_{t=1}^{T-1} \left\| W(\mathbf{F}_t; \mathbf{Q}_t, \widehat{\mathbf{Q}}_t) - W(\mathbf{F}_{t+1}; \mathbf{Q}_{t+1}, \widehat{\mathbf{Q}}_{t+1}) \right\|_2 \quad (2)$$

In this equation, the difference with Eq. 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:

$$L = (1-\lambda)L_b + \lambda L_f \quad (3)$$

In Eq. (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 changing λ during the stabilization.

Network Structure Our network structure is inspired by the 2D autoencoder network structure. However, our formulation only provides sparse feature points as 1D vectors. The input dimension does not match the 2D network structure. Moreover, the vanilla autoencoder structure 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. 5. 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. 3.1 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. 5Ⓐ. 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. 5Ⓑ. Similarly, we concatenate the 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. 5Ⓒ and Ⓓ), which only convolve with the last dimension of the tensors. The encoded tensor from different downsample levels are weighted by λ and concatenated for skip connection to decoders (Fig 5Ⓔ), 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 Design Conventional neural networks contain activation layers to introduce non-linearity. While we started with this design, we found, perhaps surprisingly, that better performance could be obtained by removing the non-linearities(supplementary material Table d). Specifically, our network does not contain activation layers, which is different from conventional neural networks. Intuitively, the definition of the loss function(Eq. 3) requires a linear relationship between the input and the output of the stabilization network, i.e. N times larger feature point coordinates require N times larger output displacement that compensates the motion.

An obvious question to ask here is why training a network is necessary to represent the linear relationship. In principle, we could pose the problem as an optimization in two alternative ways. First, it can be modeled as a linear problem in which we solve for a matrix that linearly transforms the vector of input feature points into the output displacement vector. However, this approach leads to an underdetermined problem with too many variables to be solved for in the full matrix. Second, we can use a non-linear solver to directly optimize the loss function by solving for the output displacement vector. However, this solution is prohibitive due to the runtime performance and result quality.

In the supplementary material, we provide a more thorough analysis of our choice of using a linear network. Briefly, the linear neural network factorizes or regularizes the full matrix optimization (first alternative solution) into smaller sub-problems that are easier to solve with fewer variables. Specifically, our analysis includes the necessity of using a network(Sec. B.2), why posing the problem as a non-linear optimization is prohibitive(Sec. B.3) and the performance comparison with traditional neural networks(Sec. B.4).

Training Our dataset does not contain ground truth stable videos. Therefore, our training procedure is unsupervised. The goal is to learn to minimize the loss function defined in Eq. 3, i.e. the distances between feature points/face vertices detected in consecutive frames. Note that the warping is learned solely from groups of unstructured feature points/face vertices. To avoid overfitting, we need sufficient diversity in the spatial distribution of these points and motion patterns across the frames. Previously discussed efforts we made to satisfy this requirement include a large selfie video dataset(Sec. 4.1) and randomly drawn feature points/face vertices(Sec. 4.3). In addition, we further perturb

Figure 6. *Part of the 25 selfie video examples referred to in Sec. 6. Please find complete video stills and corresponding IDs in the supplementary material. Our example videos are selected to cover a variety of challenging scenarios in real applications.*

the coordinates of feature points/face vertices using a 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 $(0, 1)$. We use Adam optimizer with a 10^{-4} learning rate to minimize the loss (Eq. 3) on length T selfie video clips randomly drawn from our dataset.

5. Warping Acceleration

As discussed in Sec. 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 $\mathbf{G} \in \mathbb{R}^{2\times 4}$: $\mathbf{v} = \mathbf{GD}$, 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 $\widehat{\mathbf{Q}}_t$: $\widehat{\mathbf{g}}_j = W(\mathbf{g}_j; \mathbf{Q}_t, \widehat{\mathbf{Q}}_t)$. 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: $\widehat{\mathbf{v}} = \widehat{\mathbf{G}}\mathbf{D}$, 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 grid approximation is negligible. Our method is not sensitive to the selection of the grid size. In our experiment, we use a grid 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.

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 that contain a variety of challenging scenarios in real applications, and are completely separate from our training dataset. Part of the testing examples are shown in Fig. 6. The complete example video stills with video IDs will be provided in supplementary Fig. c. 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. 2, we also provide visual and quantitative comparison with the offline selfie video stabilization [24], the real-time selfie video stabilization Steadiface [19], the real-time general video stabilization MeshFlow [13], the offline general video stabilization bundled camera paths [14] and the state-of-the-art learning-based methods [4] and [21]. Since our videos do not contain gyroscope data, we compare with Steadiface [19] using only the examples provided in their paper. Apart from the results discussed in this section, we provide more discussion regarding the number of feature points(warp nodes) in supplementary Sec. C.2, ablation study regarding the FG/BG mask in supplementary Sec. C.3 and performance with different input resolution in supplementary Sec. C.4.

6.1. Value of λ

In Fig. 7 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 region is, the more stable it is in the 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.

6.2. Visual Comparison

We show sample frames from our examples and the stabilized results in Fig. 8. Our method stabilizes the frames without introducing visual distortions. The real-time general video stabilization method [13] and offline general video stabilization method [14] 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 [13], since the motion within a mesh cell can be significantly different due to the foreground occlusion. The learning based method [21] 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 [4] 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 [24]. However,

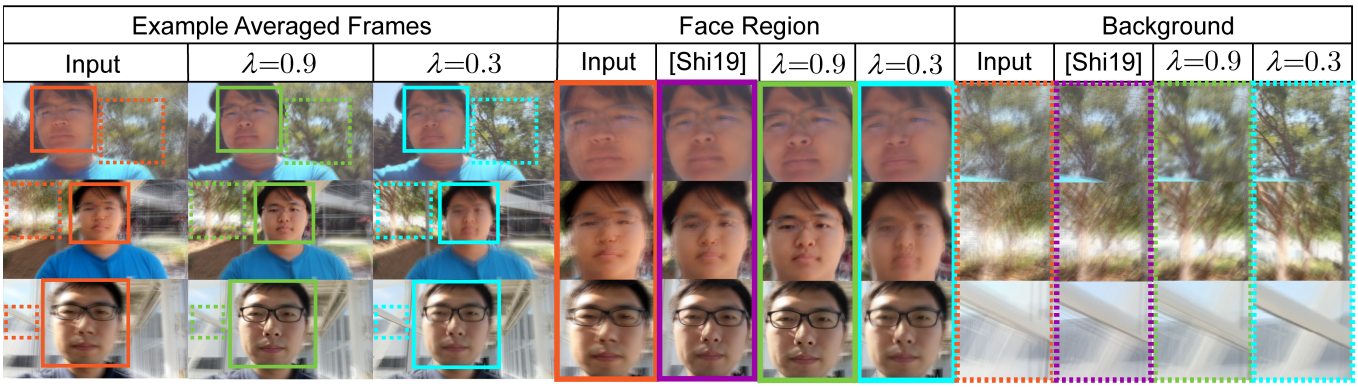

Figure 7. *The visual comparison of different values of λ in our method and the state-of-the-art real-time face stabilizaiton method Steadiface [19] using the example videos provided in their work. The images shown are the average of 15 consecutive frames. The face regions and the background regions of the input, the corresponding regions of Steadiface [19] and our method are shown in the insets on the right.*

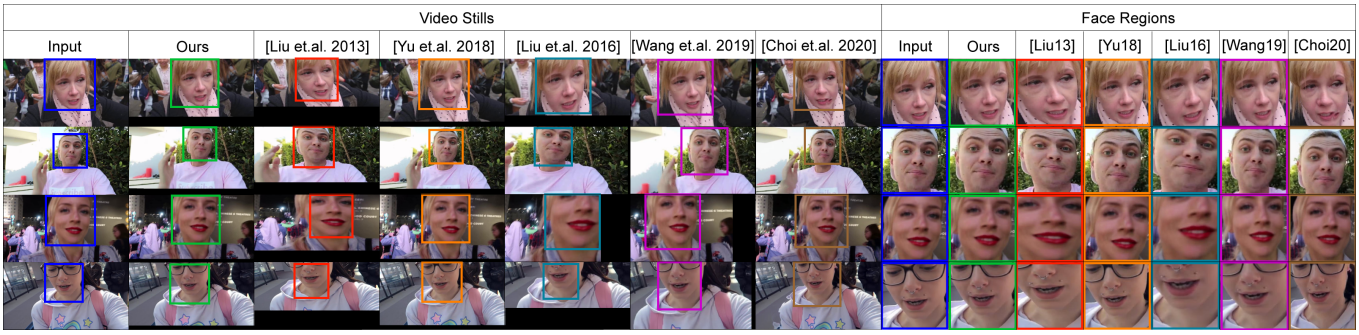

Figure 8. *The visual comparison of bundled camera paths [14], selfie video stabilization [24], MeshFlow [13], deep online video stabilization [21], deep iterative frame interpolation [4] and our method. The details of the face regions are shown in the insets on the right. We recommend readers to zoom in and observe the details in the images.*

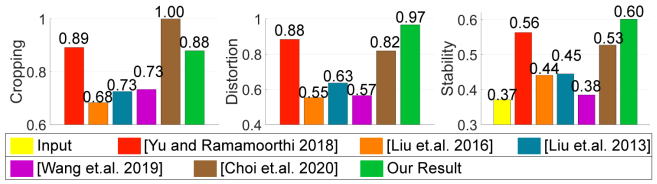

Figure 9. *Quantitative comparison of bundled camera paths [14], selfie video stabilization [24], MeshFlow [13], deep online video stabilization [21], deep iterative frame interpolation [4] and our method. In these metrics, a larger value indicates a better result. The average values over all the example videos are listed. The complete comparison on individual videos are provided in the supplementary material Fig. b.*

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. 6.4.

We also test our method on the examples in Steadiface [19], which is the state-of-the-art real-time face stabilization method. The images shown on the left of Fig. 7 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. In addition, we can alternatively set $\lambda = 0.3$ in the stabilization network. The background becomes significantly more stable than the Steadiface [19] results and our $\lambda = 0.9$ results in the averaged frames, indicating that our method is capable of stabilizing the background. Figure 7 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.

6.3. Quantitative comparison

We use the three quantitative metrics proposed in [14] to evaluate the frame size preservation (Cropping), visual distortion (Distortion) and steadiness (Stability) of the stabilization result. Note that since Steadiface [19] require gyroscope information to stabilize the video, the quantitative comparison with their method is conducted using their videos and will be discussed in Fig. 10Ⓑ.

In the left column of Fig. 9, we show the cropping metric comparison. A larger value represents a larger frame size of the stabilized result. Although [24] uses second order derivative objective, their frame size is limited by the motion of the entire video. Our sliding window 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). The frame size of our result is also significantly greater than [13], [21] and [14], since the artifacts in their results often cause over-cropping in the final video. Since [4] is based on frame interpolation, their cropping score is by default equal to 1. However, [4] 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 ours.

In the middle column of Fig. 9, 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 [13] and bundled cam-

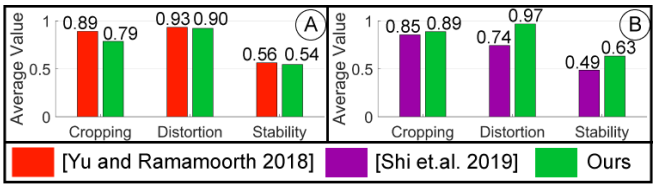

Figure 10. *Quantitative comparison with Ⓐ selfie video stabilization [24] and Ⓑ Steadiface [19] using their datasets respectively. The average values over the entire datasets are plotted. In all the three metrics, a larger value indicates a better result.*

era paths [14] introduces unexpected local distortion to the frame, which leads to the negative impact on the distortion value. The learning based methods [21] and [4] cannot generalize to selfie videos. They also produce visual artifacts that lead to even worse distortion values comparing to optimization based methods [13, 14].

The right column of Fig. 9 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 [24] with orders of magnitude improvement in stabilization speed. We also achieve better stability than [4, 13, 14, 21], which is expected since their visual result is not satisfactory as shown in Fig. 8.

To further verify the performance of our method, we also test our method on the selfie videos provided in [24] and [19]. Figure 10 shows the average values of the three metrics above on the selfie video dataset proposed by Ⓐ [24] and Ⓑ [19]. Again, our result has a quantitative performance comparable with [24]. Our method also performs better than [19] without using the gyroscope information.

6.4. Stabilization Speed

Our code is written in Python and runs on a desktop computer with an NVIDIA 2080Ti graphics card. On average, our method uses 38ms to stabilize a frame of resolution 832x448, which is equivalent to 26fps. 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 other video resolutions, we rescale the feature points to match our frame size of 832x448. The only operation impacted is the grid warping. However, since the warping is implemented on the GPU, the difference is subtle, e.g. 4ms for HD(1280x720) and 6ms for FHD(1920x1080). The overall speed is around 40ms/frame for HD and 42ms/frame for FHD. With frame size 832x448, the average stabilization time of the comparison methods(per frame) are: 4720ms for selfie video stabilization [24], 392ms for bundled camera paths[14], 8ms for Steadiface[19], 20ms for MeshFlow[13], 28ms for deep online video stabilization[21], 67ms for deep iterative frame interpolation[4]. Our method is nearly two orders of magnitude faster than the previous selfie video stabilization [24], and nearly an order of magnitude faster than the traditional optimization based general video stabilization [14]. Our method is also nearly two times faster than the frame interpolation method [4], since their network involves 2D convolutions. Also note that [4] is an offline method requiring future frames and multiple iterations through the entire video.

Although our method is slightly slower than MeshFlow [13] and deep multi-grid warping [21], we have shown in Sec. 6.2, Sec. 6.3 and supplementary video that our method produces significantly better results than theirs. Our method is also slower than Steadiface [19]. 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. In addition, since gyroscope information does not provide direct image domain motion, our approach usually yields visually more stable results as we will show in our supplementary video. As we discussed earlier in Sec. 2, our method essentially more accurately models the frame motion than Steadiface [19]. 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.

6.5. 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 [5, 6, 10, 11, 14]. In our method, this can be solved by replacing the feature tracking with the optical flow algorithm with appropriate accuracy and real-time performance.

7. Conclusions and Future Work

In this paper, 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.

Acknowledgements. This work was funded by a Qualcomm FMA Fellowship. We also acknowledge support from the Ronald L. Graham chair and the UC San Diego Center for Visual Computing.

References

[1] 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.

[2] 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.

[3] Bao Xin Chen and John K Tsotsos. Fast visual object tracking with rotated bounding boxes. In *arXiv:1907.03892*, 2019.

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

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

[6] Matthias Grundmann, Vivek Kwatra, and Irfan Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

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

[8] V. Lempitsky, A. Vedaldi, and D. Ulyanov. Deep image prior. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[9] 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.

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

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

[12] 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.

[13] 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.

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

[15] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Steadyflow: Spatially smooth optical flow for video stabilization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

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

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

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

[19] 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.

[20] Jianbo Shi and Carlo Tomasi. Good features to track. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[21] M. Wang, G. Yang, J. Lin, S. Zhang, A. Shamir, S. Lu, and S. Hu. Deep online video stabilization with multi-grid warping transformation learning. *IEEE Transactions on Image Processing*, 28(5):2283–2292, 2019.

[22] 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.

[23] Sen-Zhe Xu, Jun Hu, Miao Wang, Tai-Jiang Mu, and Shi-Min Hu. Deep Video Stabilization Using Adversarial Networks. *Computer Graphics Forum*, 2018.

[24] Jiyang Yu and Ravi Ramamoorthi. Selfie video stabilization. In *European Conference on Computer Vision (ECCV)*, 2018.

[25] Jiyang Yu and Ravi Ramamoorthi. Robust video stabilization by optimization in cnn weight space. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[26] 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.