

# Supplementary: Self-supervised Real-time Video Stabilization

Jinsoo Choi<sup>1</sup>  
jinsc37@kaist.ac.kr

Jaesik Park<sup>2</sup>  
jaesik.park@postech.ac.kr

In So Kweon<sup>1</sup>  
iskweon77@kaist.ac.kr

<sup>1</sup> KAIST  
Republic of Korea

<sup>2</sup> POSTECH  
Republic of Korea

## 1 Network details

We provide details on the Coarse-Net, Fine-Net, and Margin-Net architectures.

### 1.1 Coarse-Net

The input to the Coarse-Net is the flow map between two frames computed via the PWC-Net, which is downsampled to size  $64 \times 64$ .

| Layer name      | filter size  | channels | stride | upscale    | activation     |
|-----------------|--------------|----------|--------|------------|----------------|
| Encoder0        | $3 \times 3$ | 16       | 1      | -          | LeakyReLU(0.2) |
| Encoder1        | $3 \times 3$ | 16       | 2      | -          | LeakyReLU(0.2) |
| Encoder2        | $3 \times 3$ | 16       | 2      | -          | LeakyReLU(0.2) |
| UpScale0        | -            | 16       | -      | $\times 2$ | -              |
| Decoder0        | $3 \times 3$ | 16       | 1      | -          | LeakyReLU(0.2) |
| Concat(w/ Enc1) | -            | 16+16    | -      | -          | -              |
| Decoder1        | $3 \times 3$ | 16       | 1      | -          | LeakyReLU(0.2) |
| UpScale1        | -            | 16       | -      | $\times 2$ | -              |
| Decoder2        | $3 \times 3$ | 2        | 1      | -          | LeakyReLU(0.2) |
| Linear0         | -            | 1000     | -      | -          | LeakyReLU(0.2) |
| Linear1         | -            | 100      | -      | -          | LeakyReLU(0.2) |
| Linear2         | -            | 3        | -      | -          | -              |

Table 1: Coarse-Net architecture details.

### 1.2 Fine-Net

The input to the Fine-Net is also the flow map between two frames computed via the PWC-Net, which is downsampled to size  $32 \times 32$ .

### 1.3 Margin-Net

The input to the Margin-Net is the stabilized frame from the Fine-Net in addition to four adjacent original frames (two on each side).

| Layer name | filter size  | channels | stride | upscale | activation     |
|------------|--------------|----------|--------|---------|----------------|
| Encoder0   | $3 \times 3$ | 32       | 2      | -       | LeakyReLU(0.2) |
| Encoder1   | $3 \times 3$ | 64       | 2      | -       | LeakyReLU(0.2) |
| Encoder2   | $3 \times 3$ | 64       | 1      | -       | LeakyReLU(0.2) |
| Decoder0   | $3 \times 3$ | 32       | 1      | -       | LeakyReLU(0.2) |
| Decoder1   | $3 \times 3$ | 16       | 1      | -       | LeakyReLU(0.2) |
| AvgPool    | $3 \times 3$ | 16       | 1      | -       | -              |
| Decoder2   | $3 \times 3$ | 2        | 1      | -       | LeakyReLU(0.2) |

Table 2: Fine-Net architecture details.

| Layer name       | filter size  | channels | stride | upscale    | activation     |
|------------------|--------------|----------|--------|------------|----------------|
| Encoder0         | $7 \times 7$ | 32       | 2      | -          | LeakyReLU(0.2) |
| Encoder1         | $5 \times 5$ | 64       | 2      | -          | LeakyReLU(0.2) |
| Encoder2         | $5 \times 5$ | 128      | 2      | -          | LeakyReLU(0.2) |
| Encoder3         | $3 \times 3$ | 128      | 2      | -          | LeakyReLU(0.2) |
| Encoder4         | $3 \times 3$ | 128      | 2      | -          | LeakyReLU(0.2) |
| Encoder5         | $3 \times 3$ | 128      | 2      | -          | LeakyReLU(0.2) |
| UpScale0         | -            | 128      | -      | $\times 2$ | -              |
| Concat(w/ Enc4)  | -            | 128+128  | -      | -          | -              |
| Decoder0         | $3 \times 3$ | 128      | 1      | -          | LeakyReLU(0.2) |
| UpScale1         | -            | 128      | -      | $\times 2$ | -              |
| Concat(w/ Enc3)  | -            | 128+128  | -      | -          | -              |
| Decoder1         | $3 \times 3$ | 128      | 1      | -          | LeakyReLU(0.2) |
| UpScale2         | -            | 128      | -      | $\times 2$ | -              |
| Concat(w/ Enc2)  | -            | 128+128  | -      | -          | -              |
| Decoder2         | $3 \times 3$ | 128      | 1      | -          | LeakyReLU(0.2) |
| UpScale3         | -            | 128      | -      | $\times 2$ | -              |
| Concat(w/ Enc1)  | -            | 128+64   | -      | -          | -              |
| Decoder3         | $3 \times 3$ | 128      | 1      | -          | LeakyReLU(0.2) |
| UpScale4         | -            | 128      | -      | $\times 2$ | -              |
| Concat(w/ Enc0)  | -            | 128+32   | -      | -          | -              |
| Decoder4         | $3 \times 3$ | 64       | 1      | -          | LeakyReLU(0.2) |
| UpScale5         | -            | 64       | -      | $\times 2$ | -              |
| Concat(w/ input) | -            | 64+15    | -      | -          | -              |
| Decoder5         | $3 \times 3$ | 3        | 1      | -          | LeakyReLU(0.2) |

Table 3: Margin-Net architecture details.

## 2 Implementation Details

### 2.1 Training and testing

Applying the losses  $\mathcal{L}_C$ ,  $\mathcal{L}_F$ , and  $\mathcal{L}_M$  each to the Coarse-Net, Fine-Net, and Margin-Net respectively, we train the entire framework on four Nvidia Titan Xp GPUs (12GB VRAM) with a learning rate of 0.0001 for two days. We train our framework on the DAVIS dataset [5] for 300 epochs using the Adam optimizer ( $\beta_1 = 0.9$  and  $\beta_2 = 0.999$ ), with a batch size of 32. Our entire framework is written in Python using the Pytorch library.

To produce actual stabilized videos, we apply the Coarse-Net and Fine-Net as moving average filters. We set the window size  $N$  for the Coarse-Net to 15. The window size  $M$  for the Fine-Net is set to 15 as well. For frames near the beginning or end of a video that do not have enough neighboring frames for the window size, the window is dynamically reduced to use available adjacent frames.

### 2.2 Efficient implementation

**Reducing repetitive operations.** Note that for the Coarse and Fine stabilization, we can significantly reduce the number of redundant operations. For practical use, we reduce repetitive operations by reusing the calculations of the previous frames during the sliding window operations. The Coarse and Fine-Net operations can also be done in parallel. Details are

provided in our supplementary document. In summary, to produce one stabilized frame, the Coarse-Net requires the complexity of  $O(NM)$ , while the Fine-Net operation takes  $O(N)$ . Via our efficient implementation and approximations, we are able to reduce both operations to  $O(1)$ .

**Computation time.** Our approach takes approximately 24.3 ms to stabilize a 480p video frame (41 fps), where the Coarse-Net operation take 6.1 ms, the Fine-Net 10.5 ms, and the Margin-Net 7.7 ms. This is approximately ten times faster than offline methods [9]. We obtain optical flow using PWC-Net [9]. The time spent on PWC-Net is included in Coarse-Net and Fine-Net operations. PWC-Net does not heavily affect real-time performance since we are using small ( $64 \times 64$ ) images for flow estimation.<sup>1</sup> Since our network is *fully convolutional*, modern GPUs can feed-forward our network extremely fast with parallel computing capabilities. As a result, our method keeps real-time performance even with high-definition videos. As a result, the time for stabilizing 1080p video (FHD) is 24.7 ms which is almost identical to process 480p video. The time is measured with a the same workstation described in Sec. 2.1.

### 3 Metric computation details

**Cropping ratio** measures image area after cropping blank margins. Each homography between the input and result frame for all frames are calculated, and the scale component is recorded:

$$Sc = \sqrt{h_{00}^2 + h_{01}^2} \cdot \sqrt{h_{10}^2 + h_{11}^2}, \quad (1)$$

where  $Sc$  represents scale, and  $h$  are the elements of the homography matrix  $H \in \mathbb{R}^{3 \times 3}$ . The scale components are averaged across the entire video frames  $f_i$ , where  $i \in \{1, n\}$ :

$$CR = \frac{1}{n} \sum_{i=1}^n Sc_i, \quad (2)$$

where the  $CR$  is cropping ratio. A larger ratio indicates better video quality since it incorporates less cropping area.

**Distortion value** measures the anisotropic scaling of the homography between an input and result frame. Given the resulting video and the corresponding original video, a homography is computed between each corresponding frame. The ratio of the two largest eigenvalues are calculated for the affine part of the homography. The minimum ratio is chosen as the distortion value:

$$DV = \min_{i=1}^n \frac{\lambda_{1,i}}{\lambda_{0,i}}, \quad (3)$$

where  $DV$  is the distortion value,  $\lambda_{0,i}$  and  $\lambda_{1,i}$  are the first and second largest eigenvalues of  $i$ -th frame computed via the Eigenvalue decomposition of each homographies.

**Stability score** measures the stability of the transformations by analyzing transformation profile over the time. For this measurement, two 1D profiles are extracted by each building

<sup>1</sup>Using rescaled video frames for optical flow estimation has some benefits. (1) It can work for different video sizes. (2) It can reduce the computational burden. (3) Small videos are often enough to extract essential motion information from optical flow, as our predicted smoothed flow presented in Fig. ?? . The  $64 \times 64$  input image becomes  $1 \times 1 \times 32$  feature in the bottleneck of the PWC-Net. This setup naturally enforces global motion estimation, disregarding motion from small objects. We empirically verify that an image size of  $64 \times 64$  performs best.

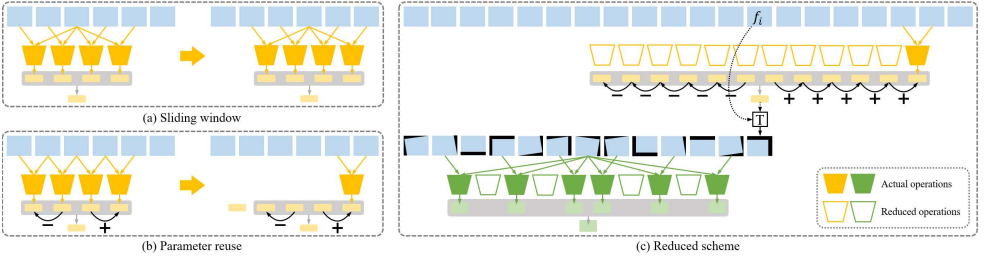


Figure 1: Reducing computational complexity via modifying the sliding window approach. The moving average filtering process of the Coarse-Net (and Fine-Net) can be thought of as the (a) sliding window process. In practice, the parameters computed between adjacent frames can be reused by adding (or subtracting) to subsequent parameter computations as shown in (b). The overall illustration of the process is shown in (c) where only one Coarse-Net operation is needed for every frame, while the Fine-Net can be sparsely computed.

a sequence of the translation and rotation components. The ratio between the sum of lowest (first to fifth) frequency energies (after Fourier transform where zeroth frequency represents the zero frequency, DC component) and the total energy is calculated:

$$S^t = \frac{\sum_{i=1}^5 c_i^t}{\sum_{i=1}^n c_i^t}, \quad S^r = \frac{\sum_{i=1}^5 c_i^r}{\sum_{i=1}^n c_i^r}, \quad (4)$$

where  $S^t$  and  $S^r$  denote energy ratios for translation and rotation profiles, and  $c_i$  represents the  $i$ -th energy coefficient. Note that the summation for the denominator is up to  $n$ , while any large number would be sufficient. The final score is determined by taking the minimum of two signals:

$$SS = \min (S^t, S^r), \quad (5)$$

where  $SS$  denotes the stability score.

## 4 Reducing repetitive operations.

Note that feed forwarding must be done  $2N = 30$  times for the Fine-Net, only after the Coarse-Net has finished stabilizing all the inputs to the Fine-Net. Thus, a sequential computation of  $2M = 30$  Coarse-Net feed-forward operations for  $2N = 30$  frames must be done, followed by  $2N = 30$  Fine-Net feed-forward operations. Thus, to produce one stabilized frame, the Coarse-Net feed-forward requires a computational complexity of  $O(M \cdot N)$ , while the Fine-Net operation takes  $O(N)$ . However, the parameter vector estimated from the Coarse-Net can be reused via the addition or subtraction of adjacent parameter vectors, as shown in Fig. 1. Thus, the Coarse-Net need only operates a feed-forward once for every subsequent  $2N = 30$  frame, reducing the complexity of  $O(M \cdot N)$  to  $O(N)$ . Note that this  $O(N)$  complexity applies only for the initial input *buffer* for the Fine-Net, and subsequent computations need only operate a feed-forward once, leading to  $O(1)$ .

In addition, we can further reduce the Fine-Net operation complexity  $O(N)$  to a constant complexity of  $O(1)$ <sup>2</sup> by fixing the number of computations within the window. It is not

<sup>2</sup>The Fine-Net operation is already of constant complexity since  $N$  is a constant, but it can be reduced to a complexity that disregards the value  $N$ , leading to  $O(1)$ .



Figure 2: Visual comparison with a real-time approach by Wang *et al.* [10]. The same content is magnified to show comparison of distortion. Our approach shows less distortion and less zoom-in effect. Please see our supplementary video for visual results.

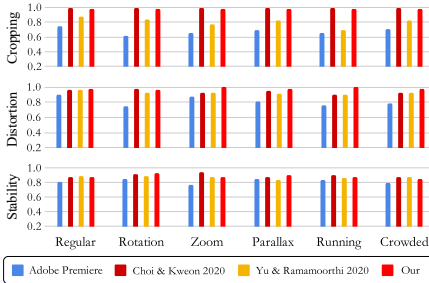


Figure 3: Metric evaluation with Adobe Premiere Pro CC [11] + two recent approaches proposed by Choi and Kweon [12] and Yu and Ramamoorthi [13] on the six challenging categories. Note that our approach runs real-time (41 frames per second) while others are not real-time approaches.

necessary to compute all  $N$  flow maps, but rather a sparse set of flow maps. For illustration, three flow maps on each side can be computed and then averaged, as shown in Fig. 1(c). Note that the Coarse-Net, Fine-Net, and Margin-Net feed-forward operations can be parallelized as well. Instead of applying the Fine-Net only after the Coarse-Net finishes its operation for the current frame  $I_i$ , the Fine-Net can operate on previous transformed frames (with one frame lag) as shown in Fig. 1(c), where the same applies to the Margin-Net as well.

## 4.1 Additional experiments

To understand the visual quality of our results, we present visual examples of the experiment shown in Fig. 2. First, we visually compare our results to the state-of-the-art real-time method of Wang *et al.* [10]. The original content are used as reference to compare the amount of distortion artifacts. Notice that our method closely resembles the original input, whereas Wang *et al.* [10] shows shape distortions. Furthermore, we can see that Wang *et al.* [10] also introduce blur and the zoom-in effect due to cropping and change in aspect ratio.

We conduct comparisons with a widely used commercial product, Adobe Premiere (Pro CC 2017), and two recent approaches by Choi and Kweon [12] and by Yu and Ramanmoorthi [13]. We first conduct quantitative evaluations on six challenging scenarios from Liu *et al.* [8], as shown in Fig. 3. We followed the same experimental protocol of previous works [8, 9, 12, 13], comparing the commercial product from Adobe Premiere on the challenging category splits of the dataset. Each category represents challenging videos where the camera exhibits quick camera motion, zooming, large parallax, and many moving people at crowded scenes. Even though Adobe Premiere and Choi and Kweon [12]’s approach run offline, and Yu and Ramanmoorthi [13] takes 570 ms per-frame, our real-time approach shows favorable results.

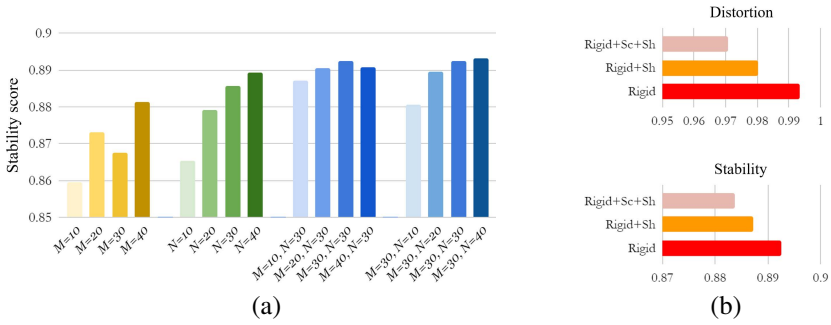


Figure 4: Ablation study on (a) window size for moving average filter  $M$  for Coarse-Net, and  $N$  for Fine-Net.  $[M = 10]$  indicates only applying Coarse stabilization with window size 10.  $[M = 10, N = 30]$  indicates Coarse-Net with window size 10 and Fine-Net with windows size 30. (b) The different configurations of image transform parameters used for Coarse-Net. Higher stability and distortion scores are better.

## 4.2 Ablation study

As an ablation study, we conduct an extensive analysis of stabilization scores when only the Coarse-Net is used, only the Fine-Net is used, and when both are used with varying window sizes. The videos used for analysis are the same as the ones used for the quantitative evaluation. Fig. 4 (a) shows the average stability scores for each setting. The cropping ratio and distortion values did not have noticeable differences for each setting, and thus we only show analysis on stability scores. The results convey that using both the Coarse-Net and Fine-Net leads to better performance. Increasing the window size generally leads to more stability with slight additive computational complexity. During the process, we found that taking a global representation of the input frames for the Coarse and Fine-Net leads to improved stabilization results. Empirically, we found that resizing the frame inputs to  $64 \times 64$  produces the best overall results.

We also conduct an ablation study among results from different number of transform parameters from the Coarse-Net. We designed the Coarse-Net to output rigid parameters (translation, rotation) as well as scale and/or shear. In Fig. 4 (b), the model producing rigid parameters, scale and shear is denoted as *Rigid+Sc+Sh* and the model producing rigid parameters and shear is denoted as *Rigid+Sh*. These models were each trained by applying random (bounded uniform distribution) rigid parameters, scale and shear transformations. The stabilized videos were generated by applying the Coarse-Net with window size  $M = 30$  and Fine-Net window size of  $N = 30$ , averaging the scores obtained from videos used in the quantitative evaluation.

Surprisingly, we observe that the *rigid* transform performed best in terms of distortion and stability scores. This suggests that the combination of rigid transformation and the smooth flow can sufficiently approximate camera motion in the wild. Another view is that scale and shear parameters may impose excess freedom on stabilization yielding poor stability. Even though the Coarse-Net utilizes a rigid transform, along with the Fine-Net, it can successfully handle various scenarios including zooming. We have confirmed its robustness against various challenging settings including zooming, parallax situations etc. as shown in Fig. 2, and 3.

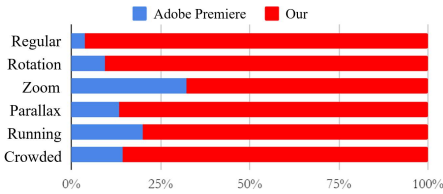


Figure 5: User study against the warp stabilizer of Adobe Premiere Pro CC. The comparisons were done on the six challenging category splits.

### 4.3 User study

For subjective evaluation, we conduct a user preference test between Adobe Premiere and our method. We recruited 35 participants with an age range of 23  $\sim$  59, where 23 were males and 12 females. The participants were asked to evaluate the videos in terms of overall quality, with an emphasis on stabilization. Fig. 5 presents the results of the user study showing that our method is more favorable. In particular, our method showed exceptional preferences for scenes with quick rotations, large parallax, and large crowds. Note that the preferences show a relatively higher response in favor of Adobe Premiere for the quick zooming category. This may be due to its content containing quick zoom, where extensive cropping may have been unnoticeable causing relatively higher response. For an in-depth ablation study regarding window sizes, please refer to our supplementary material.

## References

- [1] Stabilize with the warp stabilizer effect. <https://helpx.adobe.com/premiere-pro/using/stabilize-motion-warp-stabilizer-effect.html>. Online; accessed 5 Mar. 2020.
- [2] Jinsoo Choi and In So Kweon. Deep iterative frame interpolation for full-frame video stabilization. *ACM Transactions on Graphics (TOG)*, 39(1):1–9, 2020.
- [3] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Bundled camera paths for video stabilization. *ACM Transactions on Graphics (TOG)*, 32(4):78, 2013.
- [4] Shuaicheng Liu, Ping Tan, Lu Yuan, Jian Sun, and Bing Zeng. Meshflow: Minimum latency online video stabilization. In *European Conference on Computer Vision (ECCV)*, pages 800–815, 2016.
- [5] F. Perazzi, J. Pont-Tuset, B. McWilliams, L. Van Gool, M. Gross, and A. Sorkine-Hornung. A benchmark dataset and evaluation methodology for video object segmentation. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [6] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, 2018.
- [7] Miao Wang, Guo-Ye Yang, Jin-Kun Lin, Ariel Shamir, Song-Hai Zhang, Shao-Ping Lu, and Shi-Min Hu. Deep online video stabilization with multi-grid warping transformation learning. *IEEE Transactions on Image Processing (TIP)*, 2018.
- [8] Sen-Zhe Xu, Jun Hu, Miao Wang, Tai-Jiang Mu, and Shi-Min Hu. Deep video stabilization using adversarial networks. In *Computer Graphics Forum*, volume 37, pages 267–276, 2018.
- [9] Jiyang Yu and Ravi Ramamoorthi. Learning video stabilization using optical flow. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 8159–8167, 2020.