Fast Full-frame Video Stabilization with Iterative Optimization Supplementary Materials

Weiyue Zhao[1] Xin Li[2] Zhan Peng[1] Xianrui Luo[1] Xinyi Ye[1] Hao Lu[1] Zhiguo Cao[1*]
[1]Key Laboratory of Image Processing and Intelligent Control, Ministry of Education; School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan 430074, China
[2] Department of Computer Science, University of Albany, Albany NY 12222
{zhaoweiyue, peng_zhan, xianruiluo, xinyiye, hlu, zgcao}@hust.edu.cn
xli48@albany.edu

In this supplementary, we will expand more details that are not included in the main text due to the page limitation.

S1. Algorithm Details

We introduce 3 algorithms in the main paper. In this section, we supplement the algorithm details of the confidence map back-propagation, margin fusion approach and multi-frame fusion strategy in the main paper.

Confidence map back-propagation. Algorithm S1 summaries the strategy of confidence map back-propagation in the main paper Section 4.1. The parameters in Algorithm S1 are set to $k = 5$, $d = 10$, and $\delta_C = 0.5$.

Algorithm S1 Back-propagation for aggregated confidence map

Input: $\mathcal{Y}$: optical flow; $\mathcal{C}$: confidence map; δ_C: threshold for confidence map; d: sampling interval; k: index of the first frame;
Output: $\mathcal{M}$: updated mask containing aggregated confidence map;

- set $m_{pre} = \mathbb{1}(\hat{C}_{k+(n-1)d} - \delta_C) \in \mathcal{C}$, put m_{pre} into $\mathcal{M}$;
- **for** $i = n-1$; $i >= 0$; $i--$ **do**
  - the optical flow field $Y_{warp} = Y_{k+id} \in \mathcal{Y}$;
  - using Y_{warp} to warp m_{pre} to $\hat{m}_{pre} = Y_{warp}(m_{pre})$;
  - the binarized confidence map $m_{new} = \mathbb{1}(C_{k+id} - \delta_C)$, where $M_{k+id} \in \mathcal{C}$;
  - the final mask field $\hat{M}_{k+id} = \hat{m}_{pre} \,\&\, m_{new}$
  - put $\hat{M}_{k+id}$ into $\mathcal{M}$;
  - $m_{pre} = \hat{M}_{k+id}$
- **end for**

Margin fusion. The complete pipeline of the margin fusion approach is shown in Fig. S1. At first, we coarsely align the reference frame I^s and the target frame I^t. We then crop I^{warp} and I^t to I^s_c and I^t_c, and re-align them by the optical flow outpainting. Per Algorithm S2, we further calculate the mask M_{I^t} which indicates the chosen regions of I^t. The final result I^{result} is obtained by combining I^t, M_{I^t}, and I^{warp}_c. The parameters in the Algorithm S2 are set to $\delta_D = 0.2$, $\eta_t = 20$, and $k_{t_{in}} = 11$.

Multi-frame fusion. To adaptively determine which frame and which region should be selected, the multi-frame fusion strategy is illustrated in Algorithm S3. The parameters in the Algorithm S3 are set to $\eta_u = 25k$, $\eta_r = 1.2$, and $\eta_s = 2k$.

S2. Synthetic Dataset for Training

We proposed a model-based synthetic dataset in this paper. The settings of the homography parameters are as follows: The maximum rotation angle θ is set to $10°$. The range of scaling s is set to $0.7 \sim 1.3$. The maximum translations (d_x, d_y) in the x and y directions are 100 and 70, respectively. The maximum perspective factors in the x direction and in the y direction are 0.1 and 0.15.

For different training requirements, we apply various combinations of synthetic dataset, as shown in Fig. S2 (more visualizations can be found in the *Supplementary Video*). For camera pose regression, we use the large FOV video pair of stable and unstable. For training the flow smoothing network, we alternatively adopt small FOV video pairs, which simulate coarsely stabilized video. Aiming at the flow outpainting network, we take small-FOV stable videos for training and large-FOV for ground-truth supervising.

Data for Camera Pose Regression. For training the camera pose regression network, we need to generate unstable videos. For every frame, a random homography matrix produces an unstable frame. In practice, the perspective ef-

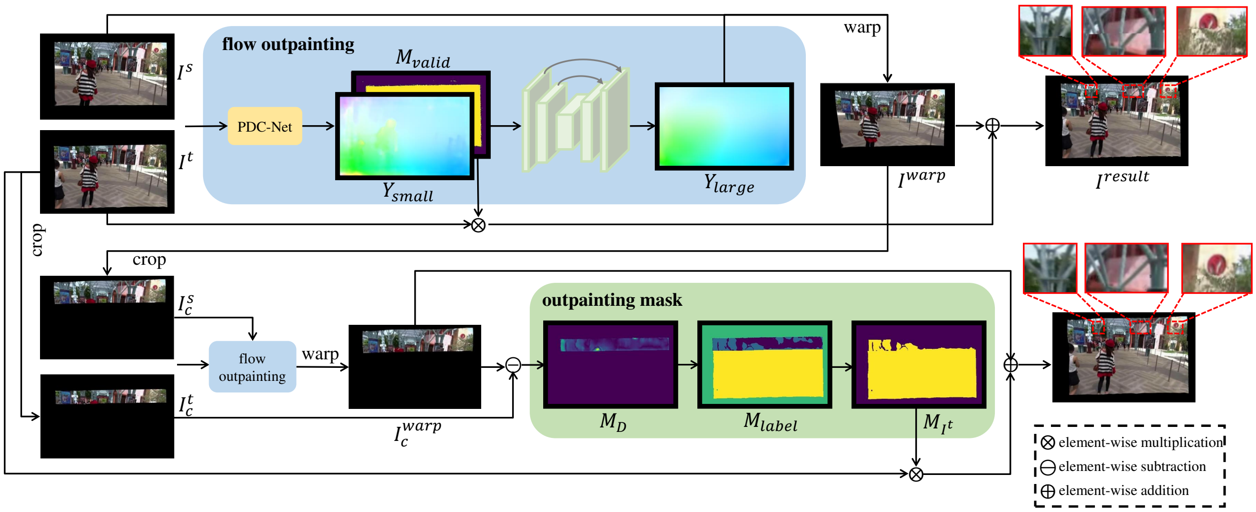

Figure S1: **Pipeline of margin fusion approach.** Given the target frame I^t, the reference frame I^s is coarsely aligned to I^{warp} by the predicted large-FOV flow field Y_{large}. Then, I^t and I^{warp} are cropped and re-aligned. Per Algorithm S2, the deduced mask M_{I^t} is fused with I^t and I_c^{warp} to obtain the resulting frame.

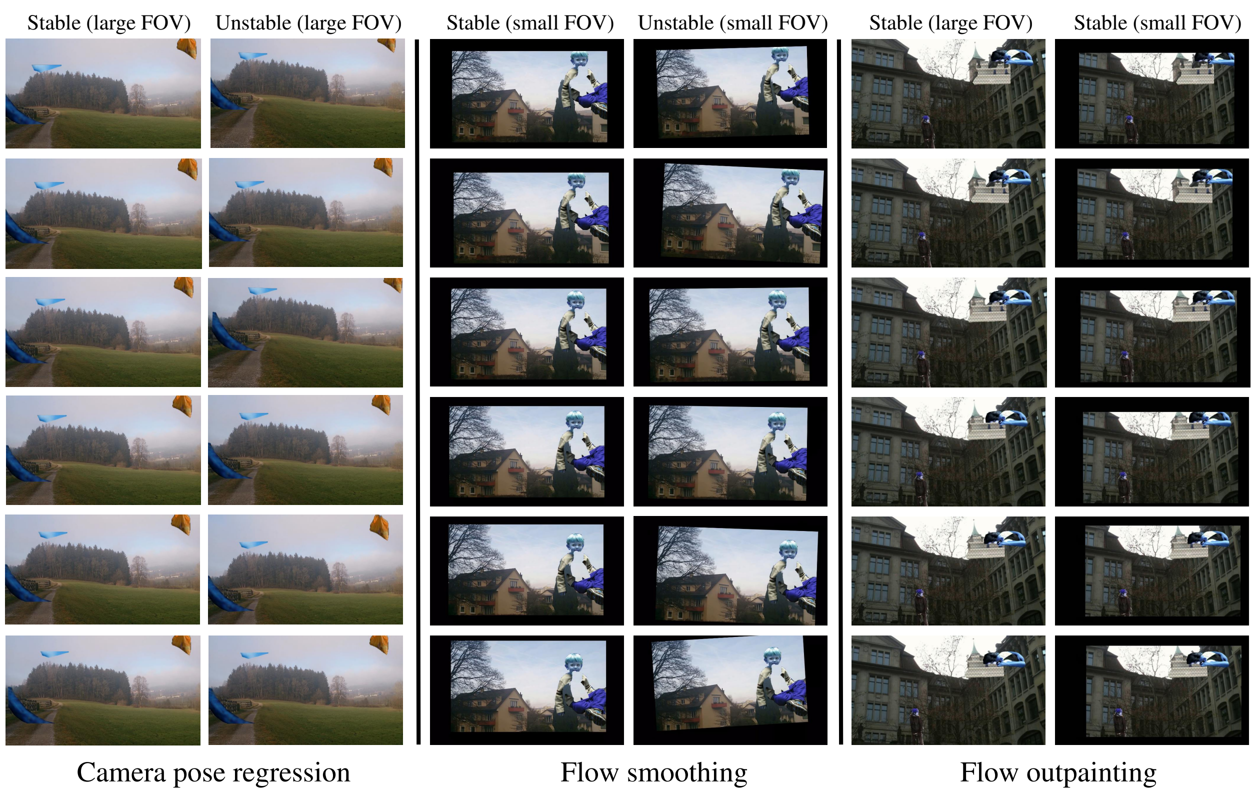

Figure S2: **Visualization of our model-based synthetic dataset.** We designed different combinations of dataset for varying tasks.

fects in the x direction and the y direction are restricted to $1e^{-5} \sim 5e^{-5}$. The pose between two unstable frames is parameterized by rotation, scaling, and translation.

Data for Flow smoothing. For training the flow smoothing network, we need to generate unstable videos with small FOV. Specifically, for the stable video, we randomly gener-

Algorithm S2 Outpainting mask Algorithm

Input: I^t: target frame; I_c^t: cropped target frame; I_c^s: cropped source frame; I_c^{warp}: warped frame of I_c^s; M^t: valid mask of I^t; M_c^t: valid mask of I_c^t;

Output: M_{I^t}: unchanged mask of I^t;

extract feature maps with VGG-16 network $f_c^t = VGG(I_c^t)$, $f_c^{warp} = VGG(I_c^{warp})$;
calculate the Euclidean distance in feature space $D = \| f_c^t - f_c^{warp} \|_2$;
$M_D = D < \delta_D$;
labeled region M_{label};
for $i, j = 0; i < h, j < w; i++, j++$ **do**
  if $\sim M^t[i,j]$ **then** $M_{label}[i,j] = 1$;
  else if $M^t[i,j] \& (\sim M_c^t[i,j])$ **then** $M_{label}[i,j] = 2$;
  else if $M^t[i,j] \& M_c^t[i,j] \& M_D[i,j]$ **then** $M_{label}[i,j] = 0$;
  else $M_{label}[i,j] = -1$;
  end if
end for
$t_{in} = M_{label}, t_{out} = 0, flag = True$;
while $\texttt{Sum}(t_{in} - t_{out}) > \eta_t$ **do**
  if flag **then**
    $t_{in} = t_{out}, flag = False$;
  end if
  inflate t_{in} with kernel size $k_{t_{in}}$, obtain $t_{out} = \texttt{inflate}(t_{in})$;
  $t_{out}[M_{label} == 1] = 1$;
  $t_{out}[M_{label} == -1] = -1$;
end while
$M_{I^t} = (t_{out} == 2)$;

ate a series of cropping mask. The cropped stable video will be jittered by random homography transformations. Then, we obtain a cropped unstable video for training and the cropped stable video for supervision.

Data for Flow Outpainting. To supervise the learning of large-FOV optical flow fields, we mask the boundaries of stable videos. Specifically, we set up a sliding window 640×360, which moves randomly with the video timeline. Then, we obtain a cropped video for training and the corresponding full-frame video for supervision.

S3. Implementation Details

We will illustrate the training details of different networks, including the camera pose regression network, the optical flow smoothing network, and the flow outpainting network. All networks are implemented using Pytorch.

Camera pose regression network. We first describe the architecture of the camera pose regression network. The network processes each input concatenated tensor $f_{in} \in \mathbb{R}^{b \times 3 \times h \times w}$ with several 2D convolutional layers, where b indicates the batch dimension and $h \times w$ indicates the spatial dimensions. The final predicted parameters are obtained by a series of 1D convolutional layers. We use a batch size of 40 and train for $10k$ iterations. we use Adam optimizer [3] with a constant leaning rate of 10^{-4} for the first $4k$ iterations, followed by an exponential decay of 0.99995 until iteration $10k$. The input resolution is set to 256×512. The weights in training loss Eq. (5) and Eq. (7) in the main paper are set to $\lambda_\theta = 1.0, \lambda_s = 1.0, \lambda_t = 1.5, \lambda_{grid} = 2.0$ for the first $6k$ iterations and $\lambda_\theta = 2.0, \lambda_s = 8.0, \lambda_t = 1.0, \lambda_{grid} = 2.0$ for the remaining $4k$ iterations.

Algorithm S3 Multi-frame Fusion Algorithm

Input: I^t: target frame; $I_{c_k}^{warp}$: warped of cropping source frame I_k^s; I_k^{result}: margin outpainting result of $I_{c_k}^{warp}$; M_k^{warp}: valid mask of $I_{c_k}^{warp}$;

Output: I^{fuse}: output fusion frame;

calculate the filling area A_k^s, misaligned region area A_k^u, and corresponding IoU ration $S_k = A_k^u/(A_k^s + 1)$ of $I_{c_k}^{warp}$;
sorted by A_k^s to obtain index list IDs;
$I^{fuse} = I^t, M^{fuse} = M_k^{warp}$;
for k in IDs **do**
  if $(A_k^u < \eta_u) \& (S_k > \eta_r) \& (A_k^s > \eta_s)$ **then**
    compute overlapped area A_k^o between $I_{c_k}^{warp}$ and I^{fuse};
    if $(A_k^o / A_k^s < \delta_r)$ **then**
      $I^{fuse} = I^{fuse} \cdot (\sim M_k^{warp}) + I_k^{result} \cdot M_k^{warp}$
    end if
  else
    `continue`;
  end if
end for

Optical flow smoothing network. We use a batch size of 6 and train for $20k$ iterations. we use Adam optimizer [3] with a constant leaning rate of 10^{-4} for the first $10k$ iterations, followed by an exponential decay of 0.99995 until iteration $20k$. The input resolution is set to 488×768.

Flow outpainting network. We apply an Unet architecture with gated convolution layers [7] as a flow-outpainting network. We use a batch size of 12 and train for $20k$ iterations. we use the Adam optimizer [3] with a constant leaning rate of 10^{-4}. The input resolution is set to 488×768. The weights in training loss Eq. (14) in the main article are set to $\lambda_{in} = 2.0, \lambda_{out} = 1.0, \lambda_F = 10.0$ for the first $10k$ iterations and $\lambda_{in} = 0.6, \lambda_{out} = 1.0, \lambda_F = 0.0$ for the remaining $10k$ iterations.

S4. Qualitative Evaluation

We show the results of the comparison of our method and the latest approaches in Fig. S3. Most methods [2, 4, 6, 9] suffer from a large amount of cropping, as indicated by the green checkerboard regions. Compared to full frame rendering approaches for interpolation [1] / generation [5], our method shows fewer visual artifacts. In particular, FuSta [5] would discard most of the input frame content for stabilization and deblurring, while we argue that video stabilization is based on destroying as little of the input frame content as possible. Thus, our method preserves the original content of the input frame as much as possible. We strongly recommend that the reviewers see our *additional supplementary video*, especially *the comparison with other full-frame approaches* (FuSta [5], DIFRINT [1]).

S5. More Experimental Results

Per-category Evaluation. We present the the average scores for the 6 categories in the NUS dataset [4].

Two-stage Stabilization. To illustrate our two-stage stabilization method, we conduct an interesting experiment. We tracked the position (x, y) of a fixed keypoint in 10 frames, where every two frames were spaced 5 frames apart. As shown in Fig. S5, the trajectory of the shaky keypoint converges to a fixed/stable position through two-stage stabilization.

Trade-off Evaluation. We have conducted the experiment to illustrate the trade-off between runtime speed and performance by varying the number of iterations in the probabilistic stabilization network and the number of input frames in the video outpainting network (as shown in the Table S1). Notably, our default configuration, consisting of 3 iterations and 7 input frames, was determined as the optimal balance between runtime speed and performance.

Table S1: The trade-off experiment between runtime speed and performance.

Iter.	Time	S.↑	Frames	Time	D.↑
1	69ms	0.80	3	57ms	0.83
2	84ms	0.84	5	78ms	0.88
3	97ms	0.86	7	97ms	0.91
4	111ms	0.86	11	132ms	0.90

Analysis of Runtime. We attribute the faster runtime of our approach against FuSta to the following three reasons: i) The traditional pose regression algorithm used in FuSta is 10 times slower than our proposed pose regression network (see Section 6.4); ii) Our method only requires computing optical flow once per frame, while FuSta requires computing it three times and relies on additional task-specific optimization and manual adjustments (see Section 6.4); iii) In the rendering stage, FuSta takes input from 11 RGB frames and their corresponding optical flow, whereas our approach only requires 7 frames. We will highlight these reasons in the final version of the manuscript.

S6. Network Architectures

Camera pose regression network. We first describe the architecture of the camera pose regression network. Given a concatenated input tensor $f_{in} \in \mathbb{R}^{3\times H\times W}$, we process it with multiple down-sampled convolution layers and flatten the output feature map to $f_{out} \in \mathbb{R}^{d\times \frac{HW}{D\times D}}$, where d, D denotes the dimension of the feature channel and the spatial down-sampling ratio, respectively. The feature vector f_{sum}, obtained by weighting the sum of f_{out} along the feature channel, regresses all parameters of the affine transformation. given by

$$w = \psi(f_{out}), f_{sum} = \sum_{i=0}^{\frac{HW}{D\times D}} w_i f_{out}(i, \cdot), \{\theta, s, d_x, d_y\} = \mho(f_{sum})\,. \tag{S1}$$

Specifically, The network processes each input concatenated tensor $f_{in} \in \mathbb{R}^{b\times 3\times h\times w}$ with several 2D convolutional layers, as shown in Table S2, where b indicates the batch dimension and $h \times w$ indicate the spatial dimensions. The final predicted parameters are obtained by a series of 1D convolutional layers.

Table S2: Modular architecture of camera pose regression network modules. Each convolution operator is followed by batch normalization and LeakyReLU (negative_slope=0.1), except for the last one. K refers to the kernel size, s denotes the stride, and p indicates the padding. We apply the Max-pooling layer to downsample each feature map.

Input Size	Convolution Layer $(K \times K, s, p)$	Output Size
Feature map extraction		
input: $b \times 3 \times h \times w$	conv0: $(3 \times 3, 1, 1)$	$b \times 8 \times h \times w$
conv0: $b \times 8 \times h \times w$	conv1: $(3 \times 3, 1, 1)$	$b \times 32 \times h \times w$
conv1: $b \times 32 \times h \times w$	pool1: $(5 \times 5, 2, 4)$	$b \times 32 \times \frac{h}{4} \times \frac{w}{4}$
pool1: $b \times 32 \times \frac{h}{4} \times \frac{w}{4}$	conv2: $(3 \times 3, 1, 1)$	$b \times 64 \times \frac{h}{4} \times \frac{w}{4}$
conv2: $b \times 64 \times \frac{h}{4} \times \frac{w}{4}$	pool2: $(5 \times 5, 2, 4)$	$b \times 64 \times \frac{h}{16} \times \frac{w}{16}$
pool2: $b \times 64 \times \frac{h}{16} \times \frac{w}{16}$	conv3: $(3 \times 3, 1, 1)$	$b \times 64 \times \frac{h}{16} \times \frac{w}{16}$
Camera pose regression		
input: $b \times 64 \times 1$	conv1: $(1, 1, 0)$	$b \times 32 \times 1$
conv1: $b \times 32 \times 1$	conv2: $(1, 1, 0)$	$b \times 16 \times 1$
conv2: $b \times 16 \times 1$	conv3: $(1, 1, 0)$	$b \times 4 \times 1$

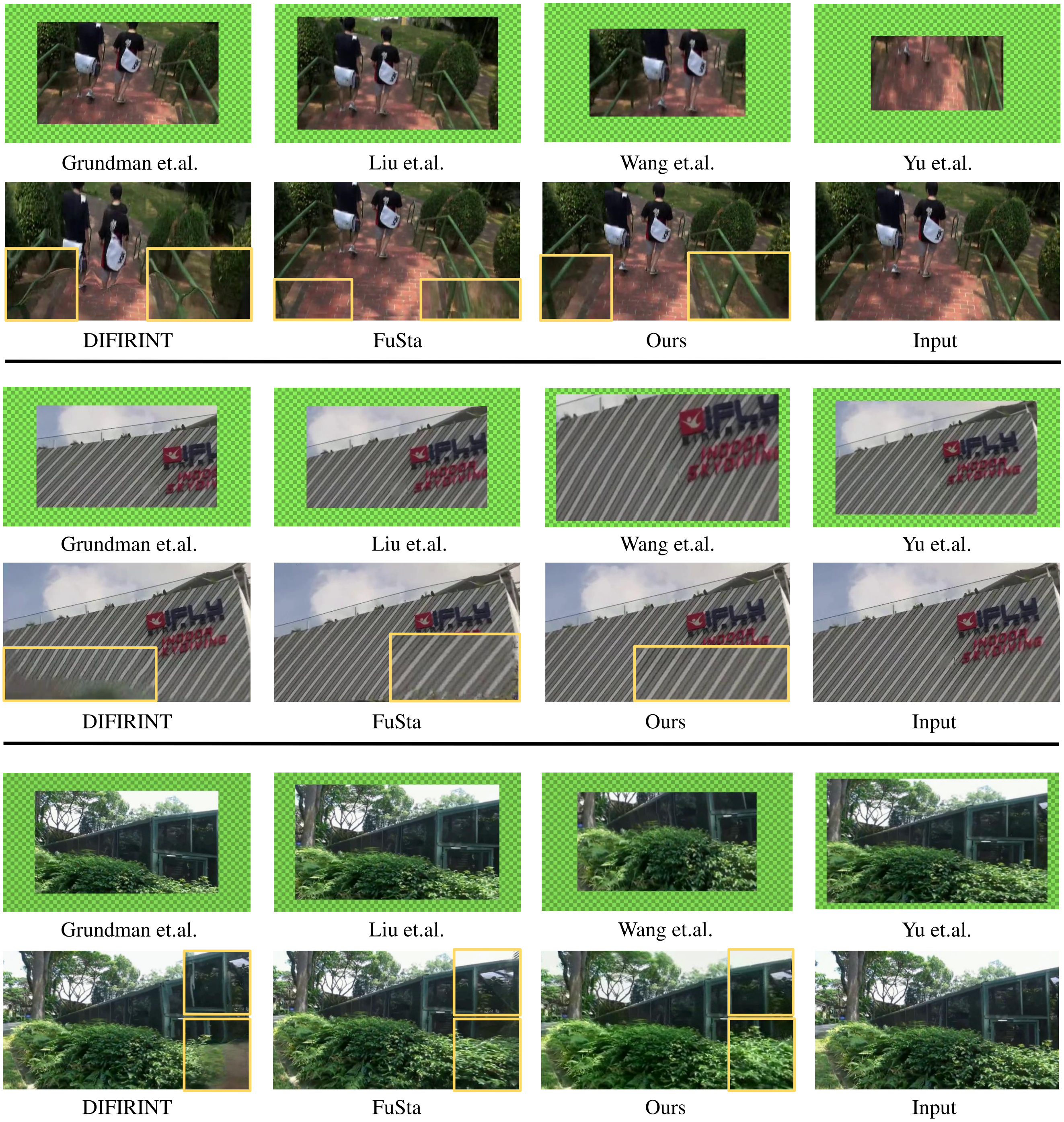

Figure S3: **Visual comparison to state-of-the-art methods.** Our proposed method does not suffer from aggressive cropping of frame borders [2, 4, 6, 9] and rendering artifacts than DIFRINT [1] and FuSta [5]. Specially, we keep more of the content in the input frames than FuSta [5].

Flow outpainting network. We apply a Unet architecture with gated convolution layers [7] as a flow outpainting network, as shown in Table S3.

S7. Limitations

Although our method achieves a comparable stability score, we use only a simple Gaussian sliding window filter

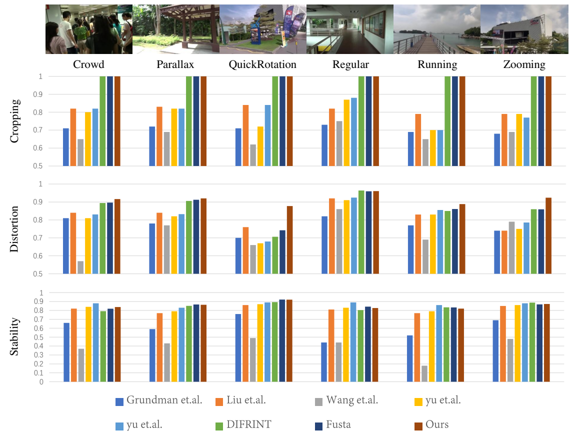

Figure S4: **Per-category quantitative evaluation on NUS dataset.** We compare the cropping ratio, distortion value, and stability score with state-of-the-art methods [2, 4, 6, 8, 9, 5, 1].

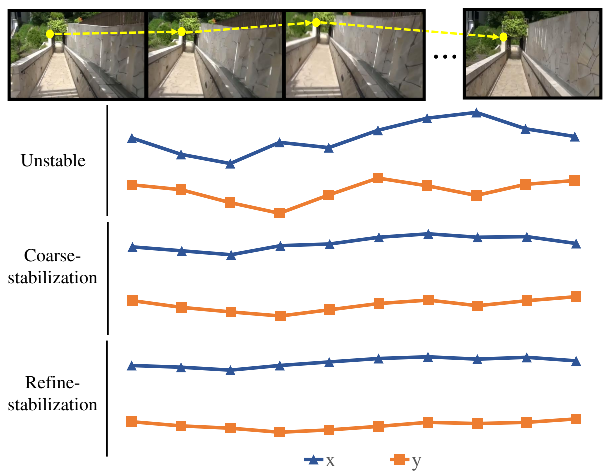

Figure S5: **Illustration of our iterative optimization-based stabilization algorithm.**

to smooth the camera trajectory in the coarse stage, leaving room for further improvement. In addition, our rendering strategy could generate artifacts in human-dense scenarios due to the nonrigid transformation of the human body, breaking our assumption of local spatial coherence.

Table S3: Architecture of the flow-outpainting network. Each 2D gated-convolution [7] ('G_conv') is followed by batch normalization and Sigmoid. The final 'conv' denotes the 2D convolution layer without batch normalization and Sigmoid. K refers to the kernel size, s denotes the stride, and p indicates the padding. We apply the Maxpooling Layer for downsampling ('down') and bilinear interpolation for upsampling ('up').

Input Size	Convolution Layer ($K \times K, s, p$)	Output Size
input: $b \times 3 \times h \times w$	down_0	$b \times 3 \times \frac{h}{4} \times \frac{w}{4}$
down_0: $b \times 3 \times \frac{h}{4} \times \frac{w}{4}$	G_conv0: $(3 \times 3, 1, 1)$	$b \times 16 \times \frac{h}{4} \times \frac{w}{4}$
G_conv0: $b \times 16 \times \frac{h}{4} \times \frac{w}{4}$	down_1	$b \times 16 \times \frac{h}{8} \times \frac{w}{8}$
down_1: $b \times 16 \times \frac{h}{8} \times \frac{w}{8}$	G_conv1: $(3 \times 3, 1, 1)$	$b \times 64 \times \frac{h}{8} \times \frac{w}{8}$
G_conv1: $b \times 64 \times \frac{h}{4} \times \frac{w}{4}$	down_2	$b \times 64 \times \frac{h}{16} \times \frac{w}{16}$
down_2: $b \times 64 \times \frac{h}{16} \times \frac{w}{16}$	G_conv2: $(3 \times 3, 1, 1)$	$b \times 64 \times \frac{h}{16} \times \frac{w}{16}$
G_conv2: $b \times 64 \times \frac{h}{16} \times \frac{w}{16}$	conv0: $(3 \times 3, 1, 1)$	$b \times 64 \times \frac{h}{16} \times \frac{w}{16}$
conv0: $b \times 64 \times \frac{h}{16} \times \frac{w}{16}$	G_conv3: $(3 \times 3, 1, 1)$	$b \times 32 \times \frac{h}{16} \times \frac{w}{16}$
G_conv3: $b \times 32 \times \frac{h}{16} \times \frac{w}{16}$	up_0	$b \times 32 \times \frac{h}{8} \times \frac{w}{8}$
up_0+G_conv1: $b \times 96 \times \frac{h}{8} \times \frac{w}{8}$	G_conv4: $(3 \times 3, 1, 1)$	$b \times 16 \times \frac{h}{8} \times \frac{w}{8}$
G_conv4: $b \times 16 \times \frac{h}{8} \times \frac{w}{8}$	up_1	$b \times 16 \times \frac{h}{4} \times \frac{w}{4}$
up_1+G_conv0: $b \times 32 \times \frac{h}{4} \times \frac{w}{4}$	conv0: $(3 \times 3, 1, 1)$	$b \times 2 \times \frac{h}{4} \times \frac{w}{4}$
conv0: $b \times 2 \times \frac{h}{4} \times \frac{w}{4}$	up_2	$b \times 2 \times h \times w$

References

[1] Jinsoo Choi and In So Kweon. Deep iterative frame interpolation for full-frame video stabilization. *ACM Trans. Graph.*, 39(1):1–9, 2020. 4, 5, 6

[2] Matthias Grundmann, Vivek Kwatra, and Irfan Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 225–232, 2011. 4, 5, 6

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. Int. Conf. Learn. Repr.*, 2014. 3

[4] Shuaicheng Liu, Lu Yuan, Ping Tan, and Jian Sun. Bundled camera paths for video stabilization. *ACM Trans. Graph.*, 32(4):1–10, 2013. 4, 5, 6

[5] Yu-Lun Liu, Wei-Sheng Lai, Ming-Hsuan Yang, Yung-Yu Chuang, and Jia-Bin Huang. Hybrid neural fusion for full-frame video stabilization. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 2279–2288, 2021. 4, 5, 6

[6] 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 Trans. Image Process.*, 28(5):2283–2292, 2019. 4, 5, 6

[7] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 4471–4480, 2019. 3, 5, 6

[8] Jiyang Yu and Ravi Ramamoorthi. Robust video stabilization by optimization in cnn weight space. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 3795–3803, 2019. 6

[9] Jiyang Yu and Ravi Ramamoorthi. Learning video stabilization using optical flow. In *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 8156–8164, 2020. 4, 5, 6