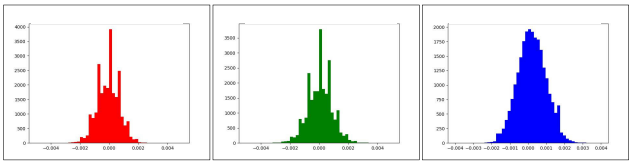
Gyro-based Neural Single Image Deblurring

Supplementary Material



(a) Noise distribution (x-axis) (b) Noise distribution (y-axis) (c) Noise distribution (z-axis)

Figure S1. Noise distributions of the Galaxy S22 gyro sensor.

S1. Overview

In the supplementary material, we provide additional analyses, implementation details and additional qualitative results on GyroBlur-Synth and GyroBlur-Real. Specifically, we provide:

- Network training details
- Details on the blur synthesis pipeline
- Additional analysis on the effect of our gyro error handling
- Details on the datasets
- Details on our extension of the non-blind deblurring methods
- Network architectures
- Additional qualitative results including real-world images with moving objects

S2. Network Training Details

Generating erroneous camera motion field When generating camera motion fields, we add gyro sensor noise and random rotational center shift to simulate real-world gyro noise. To measure gyro sensor noise distributions, we placed a Samsung Galaxy S22 smartphone on a stationary table. We then collected gyro data for 120 seconds and estimated the noise distribution for the x, y and z axes respectively. We found out that noise in the gyro data for each axis follows distinct normal distributions (Fig. S1), characterized by their respective means and standard deviations. The estimated noise distributions are:

$$n_x \sim \mathcal{N}(-0.00005643153, 0.0008631607^2) \quad (1)$$

$$n_y \sim \mathcal{N}(-0.00006369004, 0.0015023947^2) \quad (2)$$

$$n_z \sim \mathcal{N}(0.00021379517, 0.0007655643^2) \quad (3)$$

where n_x, n_y and n_z represent the noise distributions of the x, y and z axes, respectively. The amount of rotational center shift is randomly sampled from $[-500, 500]$ pixels for both x and y axes of the image plane following Hu *et al*. [4].

Noise parameter	Value
$\log_2$(shot noise) at ISO 100	-10.0009938243
$\log_2$(shot noise) at ISO 1600	-9.3348824266
Slope of $\log_2$(shot noise) - $\log_2$(read noise)	3.15578751
Intercept of $\log_2$(shot noise) - $\log_2$(read noise)	10.0003514152

Table S1. Noise parameters for RSBlur blur synthesis pipeline.

(a) Before refine (b) After 1st refine (c) After 2nd refine

Figure S2. Error map between accurate gyro feature map and erroneous gyro feature map for visualizing gyro refinement process. (a) Error map before the gyro refinement blocks. (b) Error map after the first gyro refinement block. (c) Error map after the second gyro refinement block.

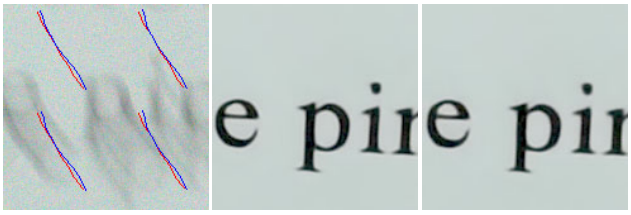
Scheduling of α To apply the curriculum-learning-based training strategy to train our network, we gradually increase α from 0 to 1 during the training. Our scheduling protocol for α is

$$\alpha = \begin{cases} 0.1 \cdot \lfloor ep/10 \rfloor & \text{if } ep < 100 \\ 1 & \text{otherwise} \end{cases} \quad (4)$$

where ep denotes the current training epoch.

S3. Blur Synthesis Pipeline

As mentioned in our main paper, we adopt the RSBlur pipeline [8] to synthesize realistic blurred images in GyroBlur-Synth. Our detailed process to generate the blurry images in GyroBlur-Synth is as follows. For generating each blurry image, we first sample a sharp image and a sequence of gyro data samples, and interpolate the gyro data samples as described in our main paper. Then, following the RSBlur pipeline, we warp the sharp image using the gyro data samples, convert the color space of the warped sharp images to the linear space, and average them to obtain a blurred image. We then perform the remaining steps of the RSBlur pipeline including the saturation synthesis, conversion to RAW, noise synthesis, and camera ISP to obtain a realistic blurred image. Regarding the shot and read noise, we estimate their distributions from a Samsung Galaxy S22 ultra-wide camera, which are reported in Tab. S1. We refer the readers to [8] for more details on the blur synthesis process.



(a) Blurry (b) No noise (c) $\sigma = 20$

Figure S3. Gyro sensor noise visualization. (a) Accurate camera motion field (Blue line) and noisy camera motion field with $\sigma = 20$ (Red line). (b) Deblurred result with accurate camera motion field. (c) Deblurred result with noisy camera motion field.

(a) Blurry (b) No error (c) Error (1000, 1000) (d) FFTformer

Figure S4. Rotational center shift visualization. (a) Accurate camera motion field (Yellow line) and erroneous camera motion field with rotational center error $(1000, 1000)$ (Red line). (b) Deblurred result with the accurate camera motion field. (c) Deblurred result with the erroneous camera motion field. (d) Deblurred result of FFTformer [6].

Noise level	0	5σ	10σ	15σ	20σ
PSNR	27.44	27.39	27.37	27.36	27.30
SSIM	0.7850	0.7843	0.7837	0.7838	0.7824

Table S2. Analysis on robustness to gyro sensor noise. σ denotes standard deviation of the noise distribution.

S4. Additional Analysis on the Effect of Our Gyro Error Handling

S4.1. Visualization of the Effect of Gyro Refinement

To see the effect of the gyro refinement blocks, we visualize errors in gyro features by computing the difference between the gyro features of erroneous and error-free gyro data in Fig. S2. While the error map between two gyro features before passing the gyro refinement blocks (Fig. S2 (a)) shows large error value, errors get reduced as the erroneous gyro feature passes consecutive gyro refinement blocks (Fig. S2 (b) & (c)). This result indicates that gyro refinement blocks do help the network to refine erroneous gyro features and extract meaningful motion information from erroneous gyro features.

Max. shift (px.)	0	250	500	750	1000	FFTformer [6]
PSNR	27.44	27.38	27.30	26.95	26.71	26.01
SSIM	0.7850	0.7829	0.7804	0.7705	0.7614	0.7481

Table S3. Analysis on robustness to rotational center shift.

	Model	PSNR	SSIM
(a)	NAFNet [2]	25.06	0.7085
(b)	NAFNet + Camera motion field	24.57	0.6802
(c)	(b) + Curriculum learning	24.61	0.6813
(d)	Ours without curriculum learning	26.94	0.7667
(e)	Ours with curriculum learning	27.28	0.7803

Table S4. Result of training NAFNet with the concatenation of camera motion fields and blurry images.

S4.2. Robustness to Gyro Error

In this section, we analyze the robustness of our method to gyro error. To this end, we construct camera motion field variants with different amounts of gyro errors. We analyze the error robustness with two gyro error sources, which are gyro sensor noise and rotational center shift. Tab. S2 and Tab. S3 show quantitative results of GyroDeblurNet on GyroBlur-Synth with different amount of gyro errors. Tab. S2 shows the result of GyroDeblurNet with different noise level. In the table, σ denotes the standard deviation of noise distributions given in Eq. (1), Eq. (2) and Eq. (3). To see the robustness to the noise only, rotational center shift errors are not considered. Tab. S3 shows the results of GyroDeblurNet with different amounts of rotational center shift. Similarly, we do not consider gyro sensor noise to see the robustness to the rotational center shift only.

As shown in Tab. S2, GyroDeblurNet shows strong robustness to the gyro sensor noise. The results show that GyroDeblurNet can be applied to sensor data with higher sensor noise without severe performance degradation. Fig. S3 demonstrates the robustness of our method to high gyro sensor noise. Tab. S3 shows that model performance gradually decreases as the amount of rotational center shift increases. However, we can also observe that GyroDeblurNet can utilize gyro data with large rotational center shift, e.g. 1000 pixels, by showing better performance than FFTformer [6] even though it is trained with gyro data whose rotational center shifts are sampled from $[-500, 500]$. Fig. S4 shows the robustness of our method to large rotational center error compared to FFTformer [6].

S4.3. Comparison against a Simple Extension of an Existing Non-gyro Method

As our approach exploits additional gyro data to deblur an image, one may wonder whether naïvely adopting gyro data would improve the performance of existing non-gyro-

based deblurring networks. Here we argue that our gyro error handling schemes including the gyro refinement and gyro deblurring blocks are crucial for effectively handling real-world gyro data, and naïvely extending a non-gyro-based method to use gyro data results in performance degradation rather than improvement.

To verify this, we conduct an experiment where we extend NAFNet [1], which is one of the non-gyro-based state-of-the-art deblurring networks, to use gyro data. Specifically, we change the first layer of NAFNet to take a concatenation of a blurred image and a camera motion field as input. Note that the modified NAFNet model takes a camera motion field of the same spatial size as the blurred image unlike GyroDeblurNet that takes a downsampled camera motion field. We then train the modified NAFNet model using the training set of GyroBlur-Synth with and without curriculum learning, and evaluate its performance on the test set of GyroBlur-Synth.

Tab. S4 shows the evaluation result. As the table shows, the modified NAFNet (Tab. S4-(b)) achieves lower PSNR and SSIM scores than the original NAFNet model despite using additional gyro data, and the curriculum learning-based training strategy does not help the model to achieve noticeable performance gain (Tab. S4-(c)). However, our method without curriculum learning achieves significantly higher PSNR and SSIM scores (Tab. S4-(d)) compared to the naïve extension of NAFNet, and our curriculum learning scheme further enhances the performance (Tab. S4-(e)). This result proves that naïve extension of a non-gyro method cannot handle erroneous gyro data, and that our gyro error handling scheme is crucial for handling real-world gyro data.

S5. Dataset Details

Synthesizing moving objects We synthesize moving objects by randomly sampling moving directions and distances. When synthesizing a blurred image with a moving object, we first randomly sample the direction of the moving object from $[0°, 360°)$. Then, we sample the moving distance of the object from the range of 30 to 70 pixels.

Post-processing GyroBlur-Real GyroBlur-Real provides both JPEG and raw DNG images. Nevertheless, in our evaluations using GyroBlur-Real in our paper, we use only raw DNG images. As our network requires 3-channel images, in our evaluation, we first demosaicked raw DNG images to use them. For demosaicking, we used the `postprocess()` function of the Python `rawpy` package.

Synchronization between the gyro sensor and the camera One well-known issue in using gyro data is that the timestamps of the gyro sensor and the camera are not synchronized. To resolve the issue when collecting the GyroBlur-Real dataset, we used the Android API that supports synchronization between the gyro sensor and the camera. Specifically, we used the Camera ITS test[1] provided by Android API to find the temporal offset between the camera and the gyro sensor at each capture. After finding the temporal offset, we used the temporal offset to compensate the temporal misalignment between the camera and the gyro sensor.

Layer	Input ch.	Output ch.	Kernel	Stride	Padding
Concat.	c, c	2c	-	-	-
GAP	2c	2c	-	-	-
Conv1	2c	c	1×1	1	0
Mul.	c	c	-	-	-
Conv2	c	c	3×3	1	1

Table S5. Detailed architecture of the gyro refinement block. GAP denotes global average pooling operation.

S6. Extension of the Non-Blind Deblurring Methods

We compared our method with non-blind deblurring approaches that are designed to handle kernel errors [7, 10]. Since they are designed for uniform blur, we extended them to handle non-uniform blur cases. Specifically, for the method of Vasu *et al*. [10], we implemented a non-blind deblurring method for non-uniform blur [3]. To apply the implemented non-blind deblurring method to the GyroBlur-Synth dataset, we converted the gyro data into patch-wise blur kernels where the size of the kernels is 160×160 with 50% overlapping area. Then, we deblurred each image in the training set of GyroBlur-Synth with regularization strengths 0.001, 0.002, 0.005 and 0.01 and trained the neural network of the method with the concatenation of the deblurred images.

For the method of Nan *et al*. [7], we first converted the blurry images into patches whose sizes are 160×160 with 50% overlapping area. Similarly, we converted the gyro data into patch-wise blur kernels whose sizes are 160×160 with 50% overlapping area. Then, we applied the method of Nan *et al*. [7] to each patch and alpha-blended the results to generate full-resolution results.

S7. Network Architecture

Tab. S5, Tab. S6, Tab. S7 and Tab. S8 show detailed architectures of the gyro refinement block, gyro deblurring block, gyro module and image deblurring module respectively.

[1] https://source.android.com/docs/compatibility/cts/camera-its-tests?#test_sensor_fusion

Layer	Input ch.	Output ch.	Kernel	Stride	Padding
Concat.	256, 256	512	-	-	-
Conv1	512	18	3×3	1	1
Deform. conv.	256	256	3×3	1	1
Spatial attn.	256	256	-	-	-
NAFBlock	256	256	-	-	-
Concat.	256, 256	512	-	-	-
Conv2	512	256	3×3	1	1

Table S6. Detailed architecture of the gyro deblurring block.

Layer	Input ch.	Output ch.	Kernel	Stride	Padding
Conv1	16	64	3×3	1	1
GRB1	64	64	-	-	-
Conv2	64	128	3×3	2	1
GRB2	128	128	-	-	-
Conv3	128	256	3×3	2	1

Table S7. Detailed network architecture of gyro module. GRB denotes the gyro refinement block.

S8. Additional Qualitative Results

Fig. S5, Fig. S6, Fig. S7 and Fig. S8 show additional qualitative results on GyroBlur-Synth. Fig. S9, Fig. S10, Fig. S11 show additional qualitative results on GyroBlur-Real-S. Fig. S12 shows additional qualitative results on real-world images with moving objects. We show our additional qualitative results with the results of Stripformer [9], FFTformer [6], EggNet [5] for comparison.

References

[1] Liangyu Chen, Xin Lu, Jie Zhang, Xiaojie Chu, and Chengpeng Chen. Hinet: Half instance normalization network for image restoration. In *Proc. CVPR*, 2021. 3

[2] Liangyu Chen, Xiaojie Chu, Xiangyu Zhang, and Jian Sun. Simple baselines for image restoration. In *Proc. ECCV*, 2022. 2

[3] Stefan Harmeling, Hirsch Michael, and Bernhard Schölkopf. Space-variant single-image blind deconvolution for removing camera shake. In *Proc. NeurIPS*, 2010. 3

[4] Zhe Hu, Lu Yuan, Stephen Lin, and Ming-Hsuan Yang. Image deblurring using smartphone inertial sensors. In *Proc. CVPR*, 2016. 1

[5] Seowon Ji, Jun-Pyo Hong, Jeongmin Lee, Seung-Jin Baek, and Sung-Jea Ko. Robust single image deblurring using gyroscope sensor. *IEEE Access*, 2021. 4

[6] Lingshun Kong, Jiangxin Dong, Jianjun Ge, Mingqiang Li, and Jinshan Pan. Efficient frequency domain-based transformers for high-quality image deblurring. In *Proc. CVPR*, 2023. 2, 4

[7] Yuesong Nan and Hui Ji. Deep learning for handling kernel/model uncertainty in image deconvolution. In *Proc. CVPR*, 2020. 3

Layer	Input ch.	Output ch.	Kernel	Stride	Padding
Conv1	3	32	3×3	1	1
NAFBlock1_1	32	32	-	-	-
NAFBlock1_2	32	32	-	-	-
Conv2	32	64	2×2	2	0
NAFBlock2_1	64	64	-	-	-
NAFBlock2_2	64	64	-	-	-
Conv3	64	128	2×2	2	0
NAFBlock3_1	128	128	-	-	-
NAFBlock3_2	128	128	-	-	-
Conv4	128	256	2×2	2	0
NAFBlock4_1	256	256	-	-	-
NAFBlock4_2	256	256	-	-	-
NAFBlock4_3	256	256	-	-	-
NAFBlock4_4	256	256	-	-	-
GDB1	256, 256	256	-	-	-
NAFBlock4_5	256	256	-	-	-
NAFBlock4_6	256	256	-	-	-
NAFBlock4_7	256	256	-	-	-
NAFBlock4_8	256	256	-	-	-
GDB2	256, 256	256	-	-	-
NAFBlock4_9	256	256	-	-	-
NAFBlock4_10	256	256	-	-	-
NAFBlock4_11	256	256	-	-	-
NAFBlock4_12	256	256	-	-	-
GDB3	256, 256	256	-	-	-
NAFBlock4_13	256	256	-	-	-
NAFBlock4_14	256	256	-	-	-
NAFBlock4_15	256	256	-	-	-
NAFBlock4_16	256	256	-	-	-
Conv5	256	512	1×1	1	0
PixelShuffle	512	128	-	-	-
Add	128, 128	128	-	-	-
NAFBlock5	128	128	-	-	-
Conv6	128	256	1×1	1	0
PixelShuffle	256	64	-	-	-
Add	64, 64	64	-	-	-
NAFBlock6	64	64	-	-	-
Conv7	64	128	1×1	1	0
PixelShuffle	128	32	-	-	-
Add	32, 32	32	-	-	-
NAFBlock7	32	32	-	-	-
Conv8	32	3	-	-	-
Add	3, 3	3	-	-	-

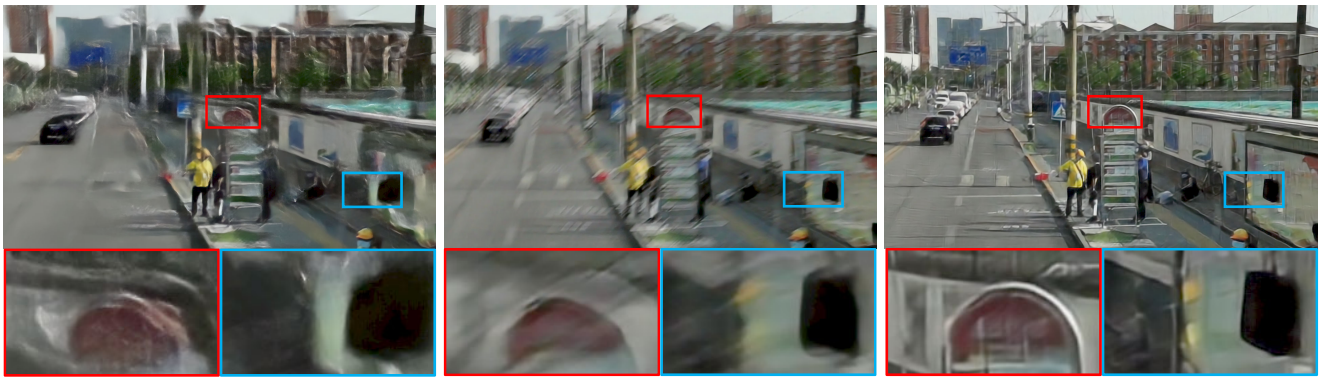
Table S8. Detailed network architecture of image deblurring module. GDB denotes the gyro deblurring block.

[8] Jaesung Rim, Geonung Kim, Jungeon Kim, Junyong Lee, Seungyong Lee, and Sunghyun Cho. Realistic blur synthesis for learning image deblurring. In *Proc. ECCV*, 2022. 1

[9] Fu-Jen Tsai, Yan-Tsung Peng, Yen-Yu Lin, Chung-Chi Tsai, and Chia-Wen Lin. Stripformer: Strip transformer for fast image deblurring. In *Proc. ECCV*, 2022. 4

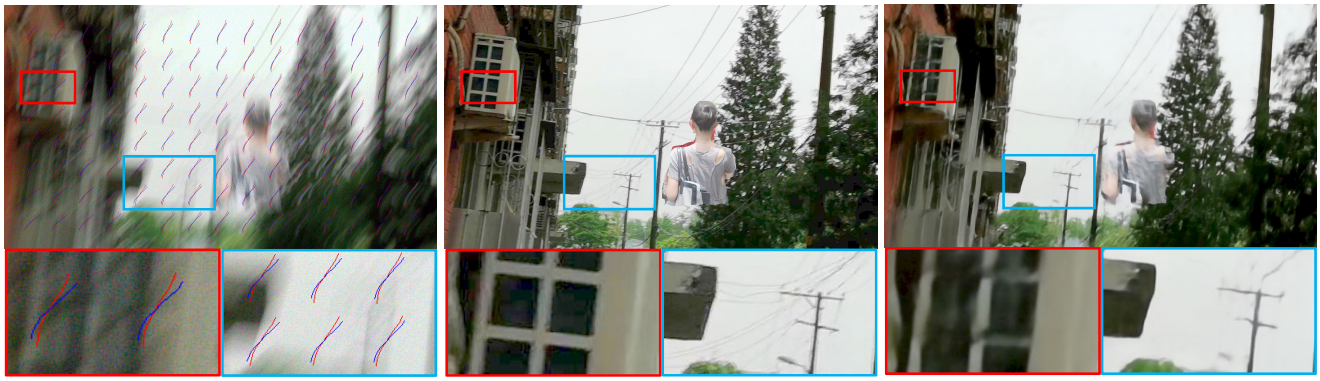
[10] Subeesh Vasu, Venkatesh Reddy Maligireddy, and AN Rajagopalan. Non-blind deblurring: Handling kernel uncertainty with cnns. In *Proc. CVPR*, 2018. 3

(a) Blurry (b) Ground-truth (c) Stripformer [9] (Single-image)



(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

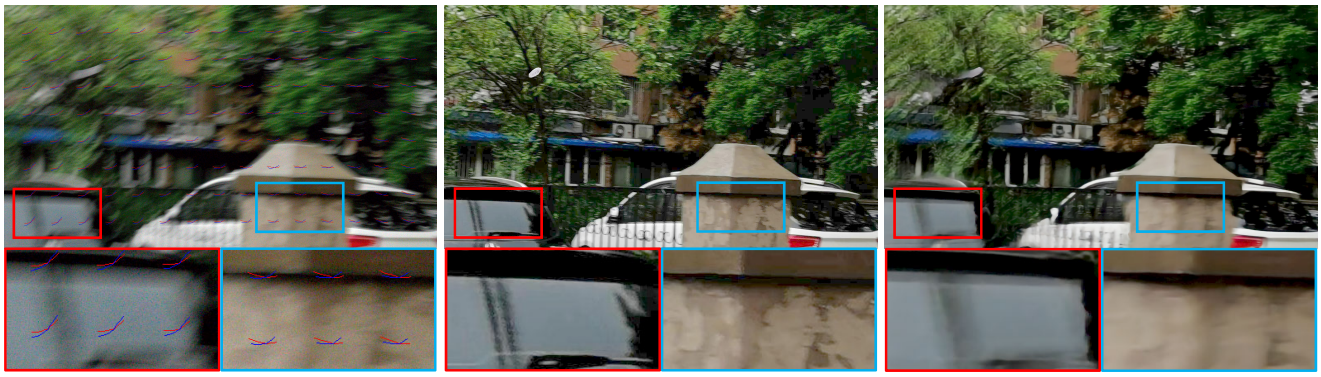
Figure S5. Additional qualitative results on GyroBlur-Synth. In (a), red lines and blue lines visualize erroneous camera motion field and accurate camera motion field respectively.



(a) Blurry (b) Ground-truth (c) Stripformer [9] (Single-image)

(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

Figure S6. Additional qualitative results on GyroBlur-Synth. In (a), red lines and blue lines visualize erroneous camera motion field and accurate camera motion field respectively.

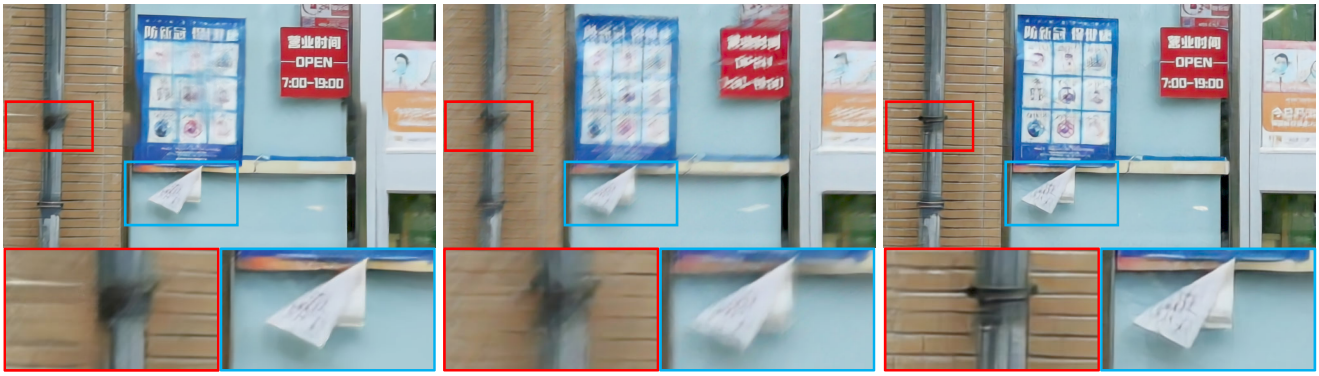


(a) Blurry (b) Ground-truth (c) Stripformer [9] (Single-image)

(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

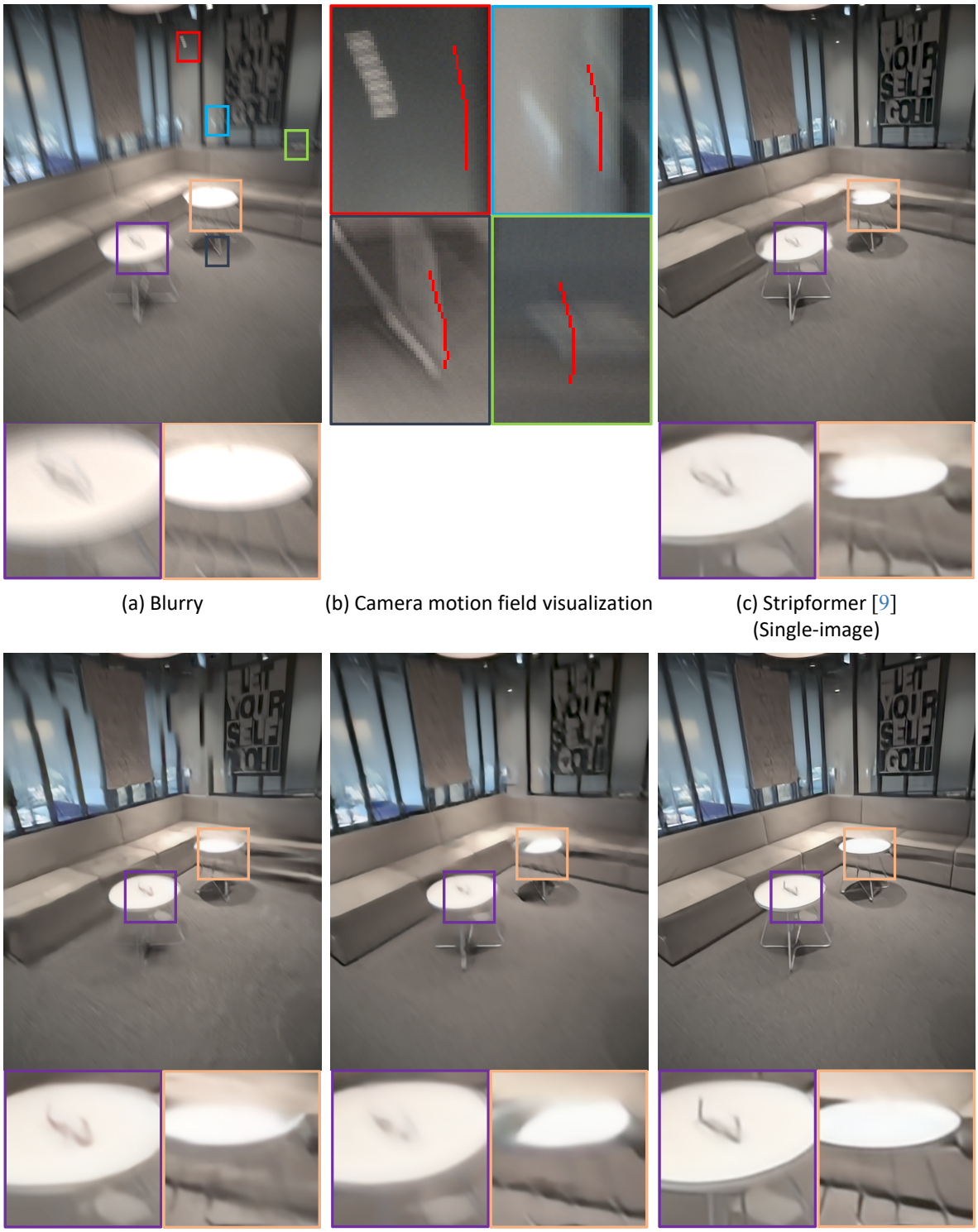
Figure S7. Additional qualitative results on GyroBlur-Synth. In (a), red lines and blue lines visualize erroneous camera motion field and accurate camera motion field respectively.

(a) Blurry (b) Ground-truth (c) Stripformer [9] (Single-image)



(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

Figure S8. Additional qualitative results on GyroBlur-Synth. In (a), red lines and blue lines visualize erroneous camera motion field and accurate camera motion field respectively.



(a) Blurry (b) Camera motion field visualization (c) Stripformer [9] (Single-image)

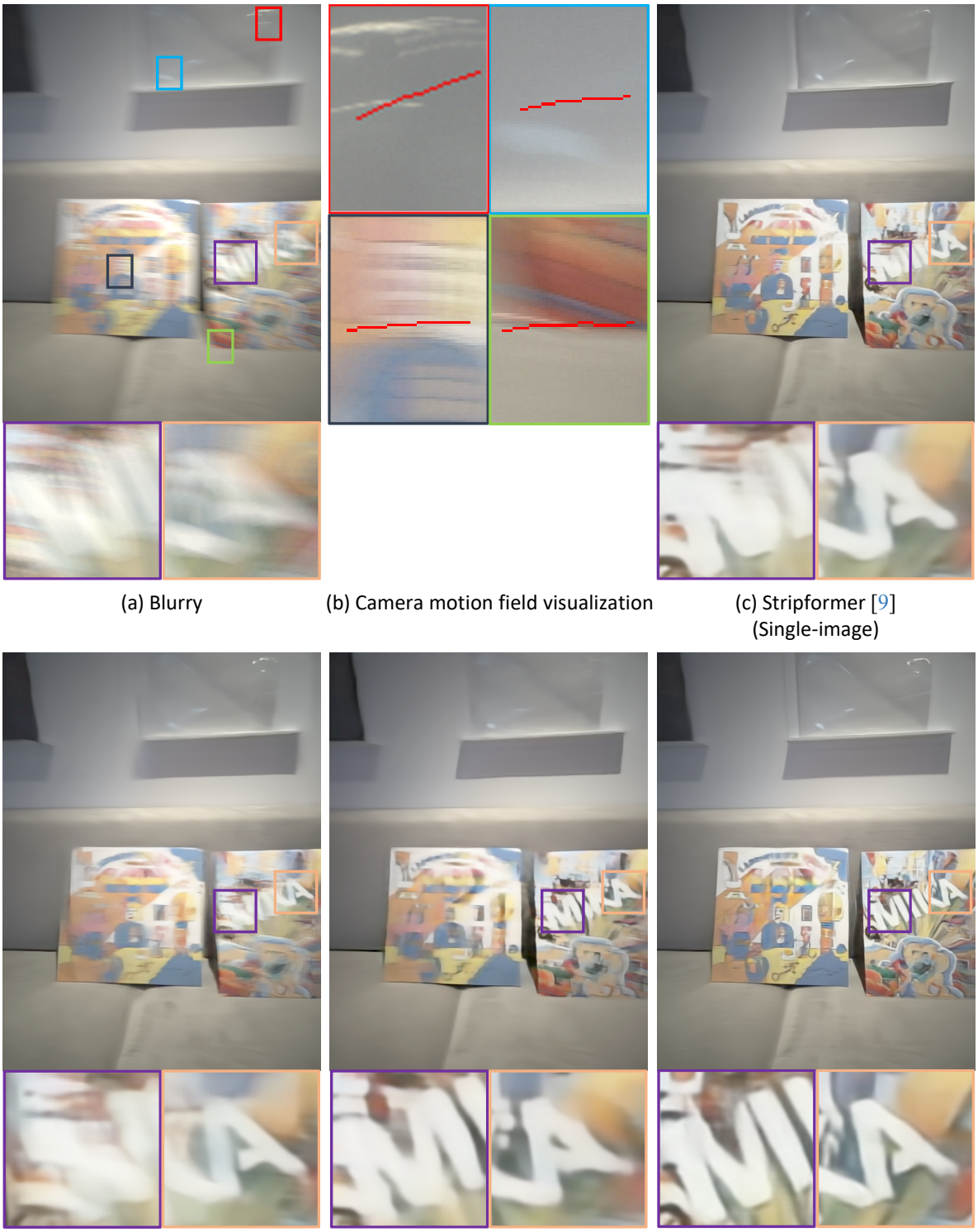
(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

Figure S9. Additional qualitative results on GyroBlur-Real. In (b), red lines visualize real-world camera motion field.

(a) Blurry (b) Camera motion field visualization (c) Stripformer [9] (Single-image)

(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

Figure S10. Additional qualitative results on GyroBlur-Real. In (b), red lines visualize real-world camera motion field.



(a) Blurry (b) Camera motion field visualization (c) Stripformer [9] (Single-image)

(d) FFTformer [6] (Single-image) (e) EggNet [5] (Gyro-based) (f) Ours (Gyro-based)

Figure S11. Additional qualitative results on GyroBlur-Real. In (b), red lines visualize real-world camera motion field.

(a) Blurry image (b) FFTformer [6] (c) EggNet [5] (d) Ours

Figure S12. Additional qualitative results on real-world image with moving objects.