Supplementary Material for Taking a Deeper Look at the Inverse Compositional Algorithm

Zhaoyang Lv[1,2] Frank Dellaert[1] James M. Rehg[1] Andreas Geiger[2]
[1]Georgia Institute of Technology, Atlanta, United States
[2]Autonomous Vision Group, MPI-IS and University of Tübingen, Germany
{zhaoyang.lv, rehg}@gatech.edu frank.dellaert@cc.gatech.edu andreas.geiger@tue.mpg.de

In this supplementary document, we first provide details regarding the network architectures used in our model. Next, we show additional experiments on the 2D affine transformation estimation task. Then we provide details on the Jacobian implementation for the rigid motion estimation. We also present more details on our TUM RGB-D experiments as well as the generation of the MovingObjects3D dataset. Finally, we provide additional qualitative experiments for our method on the MovingObjects3D dataset.

1. Network Architectures

In the following, we describe the network architectures used in our model.

(A) Two-view Feature Encoder: Fig. 1 shows the architecture of our two-view feature encoder for estimating both $\mathbf{T}_\theta$ and $\mathbf{I}_\theta$. The network takes two concatenated RGB-D views as input. For the depth channel, we use the inverse depth d clamped to [0, 10]. For the 2D affine experiments we use only the RGB channels.

The feature pyramid comprises one (A) Two-view Feature Encoder at each of the four pyramid levels. Each feature encoder uses three dilated convolutional layers. We use a Spatial Average Pooling layer to downsample the output features from the fine scale as input to the next coarser scale.

(B) Convolutional M-Estimator: Fig. 2 shows the operations and parameters of the Convolutional M-Estimator we use in this paper. In the coarse-to-fine inverse compositional refinement, we add one more input to the network which is the predicted weight from the coarser level pyramid. At each image pyramid, we bilinear upsample the weight matrix predicted from the coarse scale $\mathbf{W}_{\text{in}}$, and concatenate it with $\mathbf{T}_\theta$, $\mathbf{I}_\theta$ and $\mathbf{r}_0$, which we use as input to the Convolutional M-estimator. The network predicts $\mathbf{W}$ at the current scale. Different from traditional M-estimators which evaluate $\mathbf{W}$ at every step when $\mathbf{r}_k$ is updated, we only compute $\mathbf{W}$ once for all following K iterations. This way, we approximate the classical M-estimator and significantly reduce computation. The network is composed of four convolutional layers, with dilation [1,2,4,1], followed by a sigmoid layer which normalizes the output to the range [0,1]. Note that despite the small size of our network, the dilation layers and the coarse-to-fine process ensure a sufficiently large receptive field.

(C) Trust Region Network: Fig. 3 shows the operations and parameters of our Trust Region Network. Given the N residual maps $\mathbf{r}^k_{(i)}, i \in \{1...N\}$, we first calculate the right-hand-side (RHS) vector $\mathbf{J}^T\mathbf{W}\mathbf{r}^{(i)}_k \in \mathbb{R}^{1\times 6}$ corresponding to each residual map $\mathbf{r}^{(i)}_k$. Next, we flatten the N RHS vectors jointly with the approximate Hessian matrix $\mathbf{J}^T\mathbf{W}\mathbf{J} \in \mathbb{R}^{6\times 6}$ into a single vector, which is the input to our Trust Region Network. This network is composed of three fully connected layers and outputs the damping vector. At the last layer, a ReLU ensures non-negative elements.

2. 2D Affine Motion Estimation

The proposed framework is general and can be applied to a wide range of motion models apart from the 3D rigid motion estimation tasks presented in the main paper, see, e.g., [5]. To demonstrate its generality, this section provides results on 2D affine motion estimation. While 2D affine motion estimation is in general easier than 3D rigid motion estimation, occlusions cannot be treated explicitly as depth is unknown. Thus, any successful method must implicitly identify occlusions as outliers during estimation.

Implementation: Given pixel $\mathbf{x} = (x, y)^T \in \mathbb{R}^2$, we define the warping function $\mathbf{W}_{\boldsymbol{\xi}}$ using the following parameterization (see also [1])

$$\begin{bmatrix} 1+\xi_1 & \xi_3 & \xi_5 \\ \xi_2 & 1+\xi_4 & \xi_6 \end{bmatrix} \tag{1}$$

where $\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3, \xi_4, \xi_5, \xi_6)^T$. The analytic form of the Jacobian in (1) is

$$\begin{bmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{bmatrix} \tag{2}$$

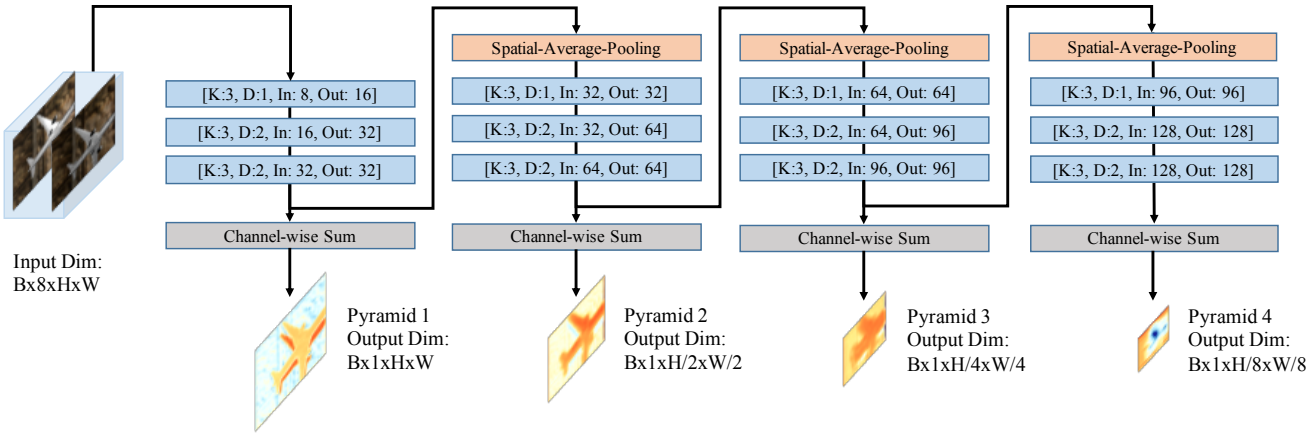

Figure 1: **(A) Two-view Feature Encoder**. We use [K, D, In, Out] as abbreviation for [Kernel size, Dilation, Input channel size, Output channel size]. All convolutional layers are followed by a BatchNorm layer and a ReLU layer. We use [B,H,W] as abbreviation for the feature size [Batch size, Height of feature, Width of feature]. We use spatial average pooling of size 2 to downsample features between two feature pyramids. We channel-wise sum the output features of the encoder at each scale to obtain the resulting feature maps.

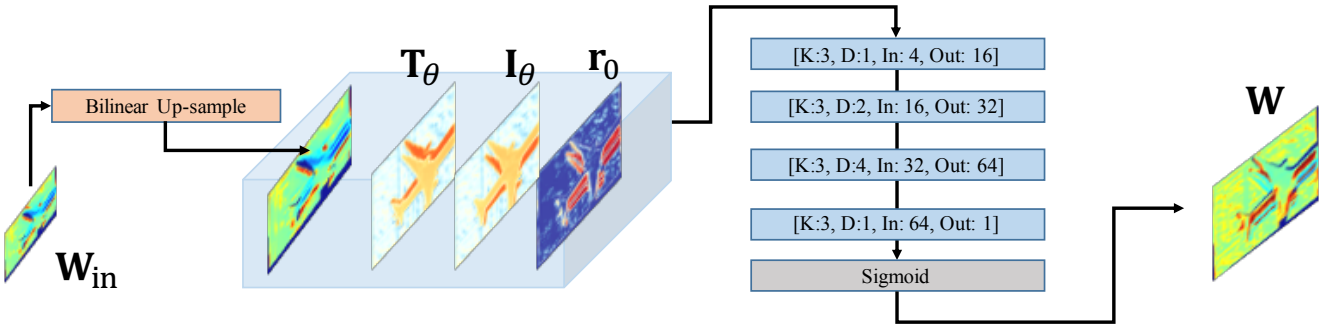

Figure 2: **(B) Convolutional M-Estimator**. We use [K, D, In, Out] as abbreviation for [Kernel size, Dilation, Input channel size, Output channel size]. All convolutional layers are followed by a BatchNorm layer and a ReLU layer. In our default weight-sharing setting, all weights are shared across networks in different pyramids. At the coarsest image level which does not require the up-sampled $\mathbf{W}$ as input, we set $\mathbf{W}_{\text{in}}$ to $\mathbf{1}$.

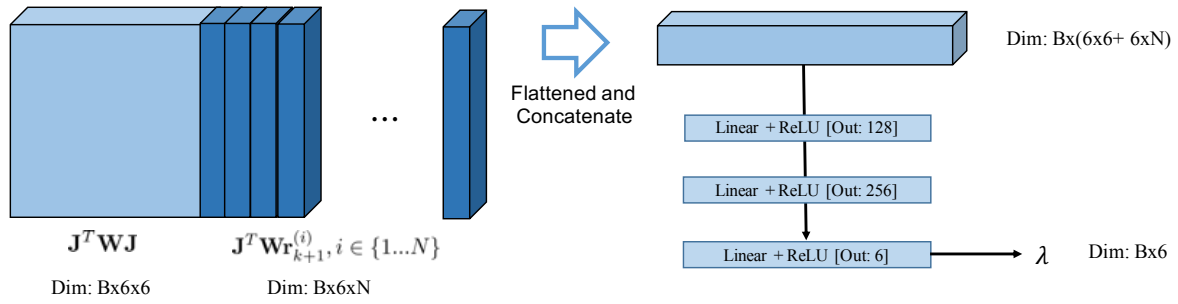

Figure 3: **(C) Trust Region Network.** We use B as abbreviation for Batch size. N indicates the number of damping proposals. In the weight-sharing setting, all weights are shared across networks at different pyramid levels. We use 'Linear' to represent a fully connected layer. The last ReLU layer ensures that the output λ is non-negative.

The composition of $\boldsymbol{\xi} \circ \Delta\boldsymbol{\xi}$ is given by

$$\begin{bmatrix} \xi_1 + \Delta\xi_1 + \xi_1\Delta\xi_1 + \xi_3\Delta\xi_2 \\ \xi_2 + \Delta\xi_2 + \xi_2\Delta\xi_1 + \xi_4\Delta\xi_2 \\ \xi_3 + \Delta\xi_3 + \xi_1\Delta\xi_3 + \xi_3\Delta\xi_4 \\ \xi_4 + \Delta\xi_4 + \xi_2\Delta\xi_3 + \xi_4\Delta\xi_4 \\ \xi_5 + \Delta\xi_5 + \xi_1\Delta\xi_5 + \xi_3\Delta\xi_6 \\ \xi_6 + \Delta\xi_6 + \xi_2\Delta\xi_5 + \xi_4\Delta\xi_6 \end{bmatrix} \tag{3}$$

The parameters of the inverse affine ξ^{-1} in (1) are

$$\frac{1}{(1+\xi_1)(1+\xi_4) - \xi_2\xi_3} \begin{bmatrix} -\xi_1 - \xi_1\xi_4 + \xi_2\xi_3 \\ -\xi_2 \\ -\xi_3 \\ -\xi_4 - \xi_1\xi_4 + \xi_2\xi_3 \\ -\xi_5 - \xi_4\xi_5 + \xi_3\xi_6 \\ -\xi_6 - \xi_1\xi_6 + \xi_2\xi_5 \end{bmatrix} \tag{4}$$

Datasets: We download 1800 natural images from flickr and use them to generate a 2D dataset for training affine 2D transformation estimation. We use the downloaded flickr images as $\mathbf{T}$. Given a random affine transform which induces a 2D warping field, we synthesize the template $\mathbf{I}$ by applying the warping field to $\mathbf{T}$ using bilinear interpolation. To remove the boundary effects caused by zero padding in the warping process, we crop a region of size 240x320 from the central region of both $\mathbf{I}$ and $\mathbf{T}$. According to the actual raw image size and the coordinates of the cropped region, we adjust the warping field to ensure the cropped region has the correct affine transform. Fig. 4 shows examples of generated pairs $\mathbf{T}$ and $\mathbf{I}$.

For each template image, we randomly generate six different affine transforms, and synthesize six different images $\mathbf{I}$ using the warping field induced from each affine transform. The generated affine transforms are used as ground truth for training and evaluation. We use five out of the six generated affine transforms and their corresponding image pairs as the training set, and use the rest for testing.

Baselines: We follow the experiment settings in the main paper. We use (A), (B), (C) to refer to our contributions in

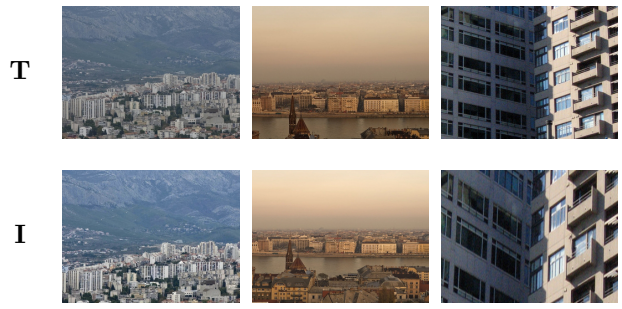

Figure 4: Examples of $\mathbf{T}$ and $\mathbf{I}$ used for our 2D affine transformation estimation experiments.

	L1 error
No Learning	0.219
DeepLK, adapted from [6]	0.158
Ours: (A)	0.151
Ours: (A)+(B)	0.132
Ours: (A)+(B)+(C)	0.071

Table 1: Quantitative evaluation on the test set using L1 error wrt. the estimated 2D affine transform.

Sec.1. We set $\mathbf{W}$ to the identity matrix when the Convolutional M-Estimator (B) is not used and use Gauss-Newton optimization in the absence of the Trust Region Network (C). We consider the following configurations:

- **Ours (A)+(B)+(C):** Our proposed method with shared weights. We perform coarse-to-fine iterations on three pyramid levels with three IC iterations at each level. We use shared weights for all iterations in (B) and (C).
- **DeepLK-6DoF** We implemented a variant of DeepLK [6] which predicts the affine transform instead of translation and scale as in the original 2D task. We use Gauss-Newton as the default optimization for this approach and no Convolutional M-Estimator. Comparing this approach with our method when using only the two-view feature network (A) demonstrates the utility of our two-view feature encoder.

Training: During training, we minimizes the L1 norm of the distance from our estimated affine transform $\boldsymbol{\xi}^\star$ to the ground truth affine transform $\boldsymbol{\xi}^{\text{GT}}$. We train our method and all baselines using a learning rate of 0.005.

Results: Table 1 shows a quantitative evaluation of our 2D affine motion estimation experiments. We evaluate the error in L1 norm by comparing the estimated results to the ground truth. Compared to all baseline methods, our model ((A)+(B)+(C)) yields the most accurate solution.

Note that different from the 3D motion estimation experiments in the main paper, there exists no motion ambiguity in the datasets used for 2D affine transform, rendering this setting simpler. We observe small improvements when using our two-view feature encoder (A) compared to using only a single view [6]. Using (B) Convolutional M-estimator gives better performance which may potentially address outliers induced by occlusions. Using (C) Trust Region Network further helps the network to boost performance. Similarly to our results in the main paper, we observe that we obtain the most accurate results when combining all components ((A)+(B)+(C)) of our model.

3. Implementation Details of Jacobian

Given pixel $\mathbf{x} \in \mathbb{R}^2$, camera intrinsic $\mathbf{K}$ and depth $D(\mathbf{x})$, we define the corresponding 3D point $\mathbf{p} = (p_x, p_y, p_z)$ as $\mathbf{p} = D(\mathbf{x})\mathbf{K}^{-1}\mathbf{x}$. At each iteration, we apply an exponential map to the output of the network $\xi \in \mathfrak{se}(3)$ to obtain the transformation matrix $\mathbf{T}_\xi = \exp([\xi]_\times)$. Suppose $\mathbf{K}$ is the intrinsic matrix for a pin-hole camera without distortion, which can be parameterized as $[f_x, f_y, c_x, c_y]$ with f_x, f_y as its focal length and c_x, c_y as its offset along the two axes. The Jacobian $\mathbf{J}$ of template image $\mathbf{T}(\mathbf{0})$ with respect to warp parameter ξ is given by

$$\mathbf{J} = \nabla\mathbf{T}\frac{\partial\mathbf{W}}{\partial\boldsymbol{\xi}} \tag{5}$$

$$\nabla\mathbf{T} = \begin{bmatrix} \frac{\partial\mathbf{T}(\mathbf{0})}{\partial u} & \frac{\partial\mathbf{T}(\mathbf{0})}{\partial v} \end{bmatrix} \tag{6}$$

$$\frac{\partial\mathbf{W}}{\partial\boldsymbol{\xi}} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \cdot \begin{bmatrix} -\frac{p_x p_y}{p_z^2} & 1+\frac{p_x^2}{p_y^2} & -\frac{p_y}{p_z} & \frac{1}{p_z} & 0 & -\frac{p_x}{p_z^2} \\ -1-\frac{p_y^2}{p_z^2} & -\frac{p_x p_y}{p_z^2} & \frac{p_x}{p_z} & 0 & \frac{1}{p_z} & -\frac{p_y}{p_z^2} \end{bmatrix} \tag{7}$$

where $\cdot$ is the element-wise product along each row. $\nabla\mathbf{T}$ is the image gradient of template $\mathbf{T}(\mathbf{0})$. In the proposed model, we compute the Jacobian of $\mathbf{T}_\theta$ from the output of network (A) by substituting $\mathbf{T}$ with $\mathbf{T}_\theta$ in (5).

We further simplify (7) by exploiting the inverse depth parameterization $\mathbf{p} = (p_u/p_d, p_v/p_d, 1/p_d)$ where $(p_u, p_v) \in \mathbb{R}^2$ is the pixel coordinate and $p_d \in \mathbb{R}^1$ is the inverse depth. We obtain:

$$\frac{\partial\mathbf{W}}{\partial\boldsymbol{\xi}} = \begin{bmatrix} f_x \\ f_y \end{bmatrix} \cdot \begin{bmatrix} -p_u p_v & 1+p_u^2 & -p_v & p_d & 0 & -p_d p_u \\ -1-p_v^2 & p_u p_v & p_u & 0 & p_d & -p_d p_v \end{bmatrix} \tag{8}$$

For more details we refer the reader to [2].

4. Details of TUM RGB-D Experiments

Our evaluation split of the TUM RGB-D SLAM dataset [4] consists of four trajectories of different conditions, trajectory length and motion magnitudes. After synchronization of the color image, depth and the ground truth trajectory, we obtain 750 frames in 'fr1/360', 584 frames in 'fr1/desk', 2203 frames in 'fr2/desk' and 830 frames in 'fr2/pioneer_360'. To better understand our method operating under different conditions, we present a detailed quantitative evaluation in Table 2 using the Relative Pose Error (RPE) and the 3D End Point Error (3D EPE) metrics for each trajectory.

Discussion: Our method with all modules (A)+(B)+(C) outperforms the baseline RGBD visual odometry [3] across all subsampled trajectories, except in 'fr1/desk' of keyframe [1,2] and 'fr2/desk' of keyframe 2 in which the performance is close to each other. Our method shows clear advantages when the motion magnitude is large, e.g. 'fr1/360', 'fr2/pioneer_360'. From the ablation, we also observe that adding the trust-region module (C) helps to stabilize training and yields the most significant improvement in test accuracy. One possible reason for this is that the trust-region module can adjust the damping parameters and adapt to a wide range of motion magnitudes in the data. At training time, adaptive damping helps to stabilize the training loss and potentially contributes to learning better features in modules (A) and (B). At inference time, the method can adjust its trust-region step to adapt to different motion magnitudes with a fixed number of iterations.

(a) aeroplane (b) boat (c) bus

(d) car (e) motorbike (f) bicycle

Figure 5: Example images from the MovingObjects3D dataset. We use 'boat' and 'motorbike' categories as test set and the others as train/validation set.

5. Details of MovingObjects3D

Background Set-up: We use a cubic box of size $6 \times 6 \times 6$ as the background layout. We download 150 background textures from TextureNinja[1] and use them as the textures for the inner side of the cube. We randomly locate the camera inside the cube with zero elevation. We randomly place four point light sources at four different directions.

Object Trajectories: For each trajectory, we randomly generate keyframe poses (we use one keyframe every ten frames). We calculate poses between keyframes using the default trajectory interpolation in Blender. We randomly locate the objects within a 1.5m radius ball around the center of the cube which ensures that the objects always move inside of the box. There is a significant number of views for which only parts of the object is visible. We exclude all frames where the entire object is outside the field of view of the camera.

Rendered Images and Ground Truth: We render the images, depth maps and instance segmentation masks using

[1] https://texture.ninja/

	mRPE: θ (Deg) $\downarrow$ / t (cm) $\downarrow$				3D EPE (cm)			
	KF 1	KF 2	KF 4	KF 8	KF 1	KF 2	KF 4	KF 8
fr1/360								
RGBD VO [3]	0.46/1.03	2.45/5.26	7.47/10.31	16.08/17.32	1.33	5.98	21.34	52.50
Ours: (A)	0.50/1.33	1.32/3.84	5.68/11.79	14.33/16.26	1.24	2.39	13.37	49.60
Ours: (A)+(B)	0.45/1.18	1.00/3.18	4.96/11.62	14.23/17.52	1.18	1.92	10.67	49.13
Ours: (A)+(B)+(C)	**0.33/0.61**	**0.49/1.20**	**2.64/2.63**	**7.24/6.64**	**1.05**	**1.21**	**2.64**	**22.40**
fr1/desk								
RGBD VO [3]	**0.43/0.69**	**0.76/1.04**	3.52/5.15	10.71/19.83	**0.59**	**0.91**	5.46	18.84
Ours: (A)	0.58/1.04	1.12/2.05	2.75/4.87	7.14/11.27	0.74	1.31	3.89	12.89
Ours: (A)+(B)	0.57/1.02	1.08/2.03	2.54/4.63	6.59/10.87	0.74	1.28	3.42	11.21
Ours: (A)+(B)+(C)	0.55/0.90	0.89/1.55	**1.58/2.59**	**4.30/7.11**	0.68	0.96	**1.74**	**6.11**
fr2/desk								
RGBD VO [3]	0.30/0.56	**0.35/0.72**	0.79/1.78	1.44/3.80	0.96	**0.93**	2.20	4.75
Ours: (A)	0.30/0.54	0.44/0.98	0.72/1.94	1.46/4.30	0.80	1.04	1.60	2.93
Ours: (A)+(B)	0.31/0.56	0.45/1.03	0.77/2.04	1.43/4.14	0.80	1.04	1.62	2.83
Ours: (A)+(B)+(C)	**0.27/0.42**	0.39/0.73	**0.61/1.37**	**1.11/2.79**	**0.73**	0.94	**1.29**	**2.09**
fr2/pioneer_360								
RGBD VO [3]	1.02/1.82	2.00/4.19	4.20/6.51	8.54/14.36	6.39	9.22	21.16	48.46
Ours: (A)	0.76/1.77	1.04/3.66	2.34/8.16	7.60/14.83	3.55	5.67	13.78	39.41
Ours: (A)+(B)	0.72/1.80	0.95/3.52	2.15/6.96	6.91/13.31	3.57	5.42	12.77	36.80
Ours: (A)+(B)+(C)	**0.65/0.85**	**0.74/1.07**	**0.98/1.79**	**2.38/6.85**	**2.76**	**3.15**	**4.46**	**13.52**
Average over trajectories mRPE and EPE								
RGBD VO [3]	0.55/1.03	1.39/2.81	3.99/5.95	9.20/13.83	2.31	4.38	12.67	31.13
Ours: (A)	0.53/1.17	0.97/2.63	2.87/6.89	7.63/12.16	1.58	2.60	8.15	26.20
Ours: (A)+(B)	0.51/1.14	0.87/2.44	2.60/6.56	7.30/11.21	1.56	2.41	7.10	24.69
Ours: (A)+(B)+(C)	**0.45/0.69**	**0.63/1.14**	**1.10/2.09**	**3.76/5.88**	**1.31**	**1.57**	**2.53**	**11.03**
Average over frames mRPE and EPE								
RGBD VO [3]	0.48/0.90	1.08/2.20	2.95/4.60	6.54/10.26	1.80	3.53	9.58	23.14
Ours: (A)	0.46/0.98	0.79/2.12	2.16/5.35	5.58/9.65	1.39	2.18	6.22	19.16
Ours: (A)+(B)	0.45/0.96	0.73/2.00	1.99/5.15	5.35/9.41	1.38	2.05	5.52	18.31
Ours: (A)+(B)+(C)	**0.39/0.59**	**0.54/0.98**	**0.91/1.83**	**2.82/4.80**	**1.16**	**1.41**	**2.18**	**8.28**

Table 2: **Detailed Results on TUM RGB-D Dataset [45].** This table shows the mean relative pose error (mRPE) on our test split of the TUM RGB-D Dataset [45]. KF denotes the size of the key frame intervals.

Blender. We show examples of objects for all six categories in Fig. 5. We use images rendered using the categories 'boat' and 'motorbike' as test set and images from the categories 'aeroplane', 'bicycle', 'bus', 'car' as the training set. We will make the dataset creation tool and the dataset itself public upon publication.

6. Qualitative Visualizations

We now demonstrate additional qualitative results on 3D rigid transformation estimation on MovingObjects3D. Note the difficulty of the task (truncation, independent background object) and the high quality of our alignments.

Visualizations of Iterative Estimation: Fig. 6 and Fig. 7 show a qualitative visualization of our method at different iterations. We warp the image $\mathbf{I}$ using the iterative estimated poses at the four coarse-to-fine pyramid scales ($\mathbf{I}(\boldsymbol{\xi}_1^\star) \rightarrow \mathbf{I}(\boldsymbol{\xi}_2^\star) \rightarrow \mathbf{I}(\boldsymbol{\xi}_3^\star) \rightarrow \mathbf{I}(\boldsymbol{\xi}_{\text{final}}^\star)$). $\mathbf{I}(\boldsymbol{\xi}_{\text{final}}^\star)$ is the final output of our method. Our results demonstrate that the proposed method is able to iteratively refine the estimation towards the global optimal solution despite the challenging scenario.

Comparison to Baselines: Fig. 8 shows a comparison of our full method to DeepLK 6DoF [6] and ablated models using only parts of the proposed modules. Our results demonstrate that the combination of all modules yields high-quality registrations.

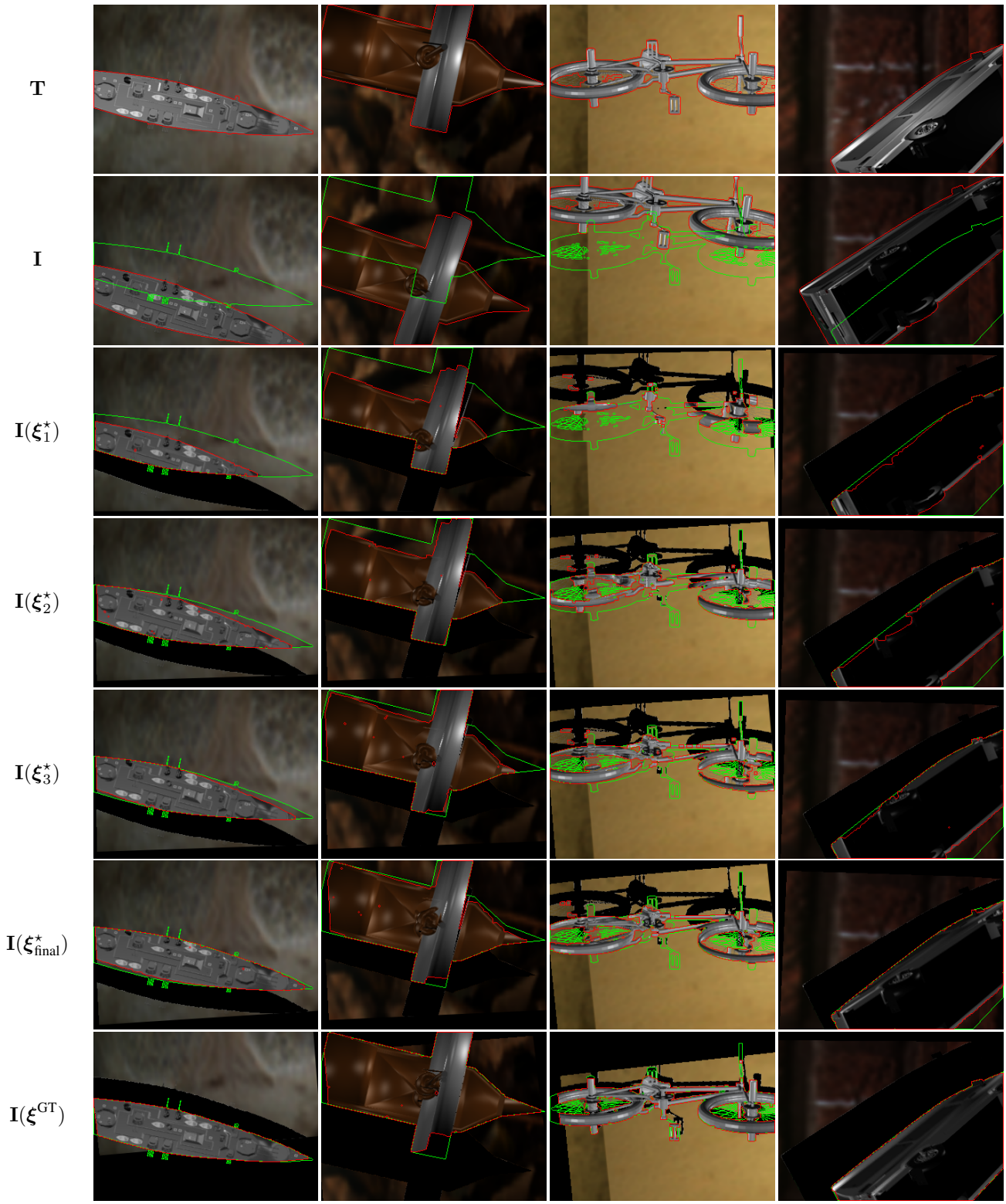

Figure 6: **Qualitative Results on MovingObjects3D.** Visualization of the warped image $\mathbf{I}(\boldsymbol{\xi})$ using the ground truth object motion $\boldsymbol{\xi}^{\text{GT}}$ (last row) and the object motion $\boldsymbol{\xi}^\star$ estimated using our method at each pyramid ($\boldsymbol{\xi}_1^\star$, $\boldsymbol{\xi}_2^\star$, $\boldsymbol{\xi}_3^\star$, $\boldsymbol{\xi}_{\text{final}}^\star$) on the MovingObjects3D *validation* and *test* set. In $\mathbf{I}$, we plot the instance boundary in red and that of $\mathbf{T}$ in green as comparison. Note the difficulty of the task (truncation, independent background object) and the high quality of our alignments. Black regions in the warped image are due to truncation or occlusion.

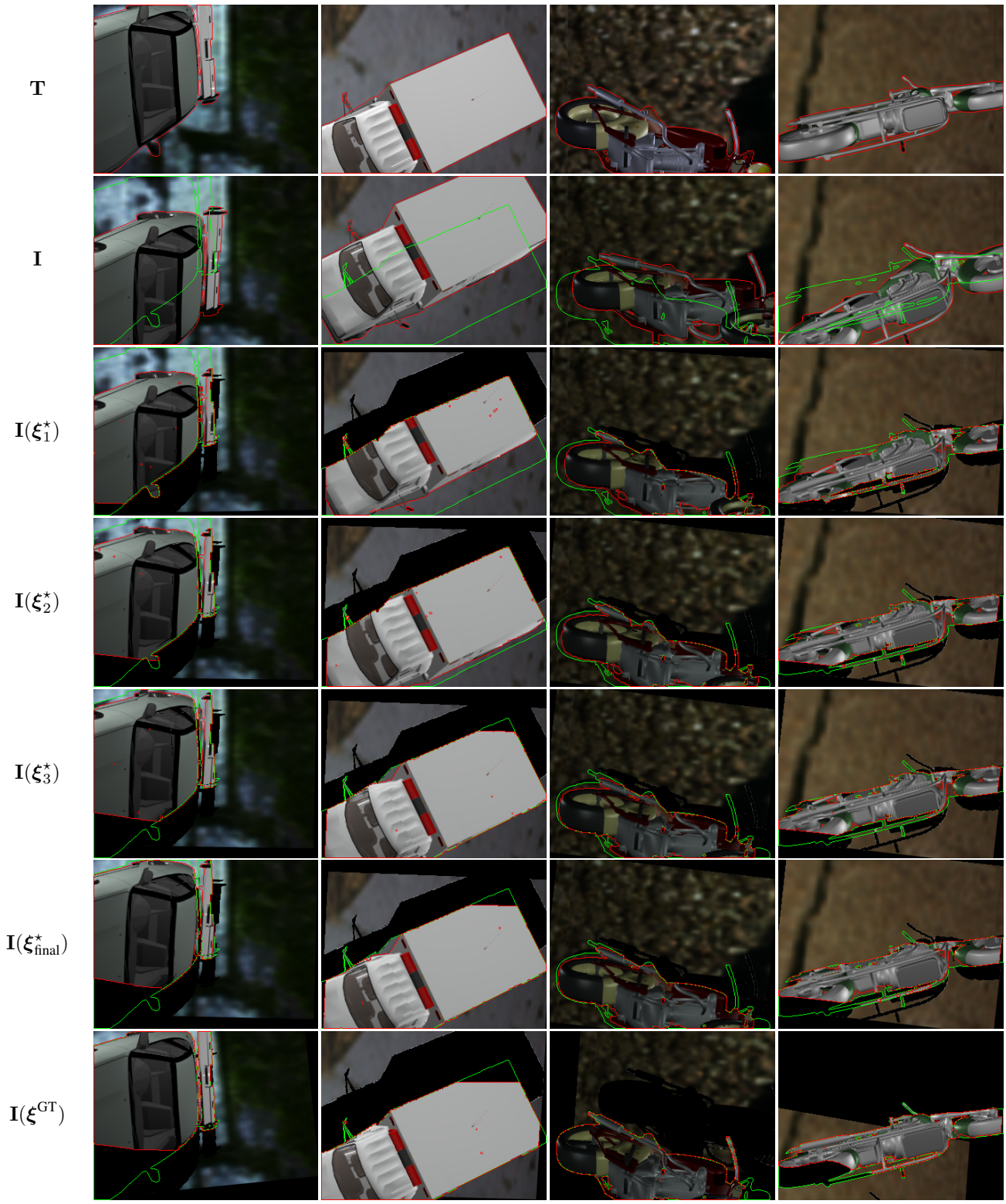

Figure 7: **Qualitative Results on MovingObjects3D.** Visualization of the warped image $\mathbf{I}(\boldsymbol{\xi})$ using the ground truth object motion $\boldsymbol{\xi}^{\text{GT}}$ (last row) and the object motion $\boldsymbol{\xi}^\star$ estimated using our method at each pyramid ($\boldsymbol{\xi}_1^\star$, $\boldsymbol{\xi}_2^\star$, $\boldsymbol{\xi}_3^\star$, $\boldsymbol{\xi}_{\text{final}}^\star$) on the MovingObjects3D *validation* and *test* set. In $\mathbf{I}$, we plot the instance boundary in red and that of $\mathbf{T}$ in green as comparison. Note the difficulty of the task (truncation, independent background object) and the high quality of our alignments. Black regions in the warped image are due to truncation or occlusion.

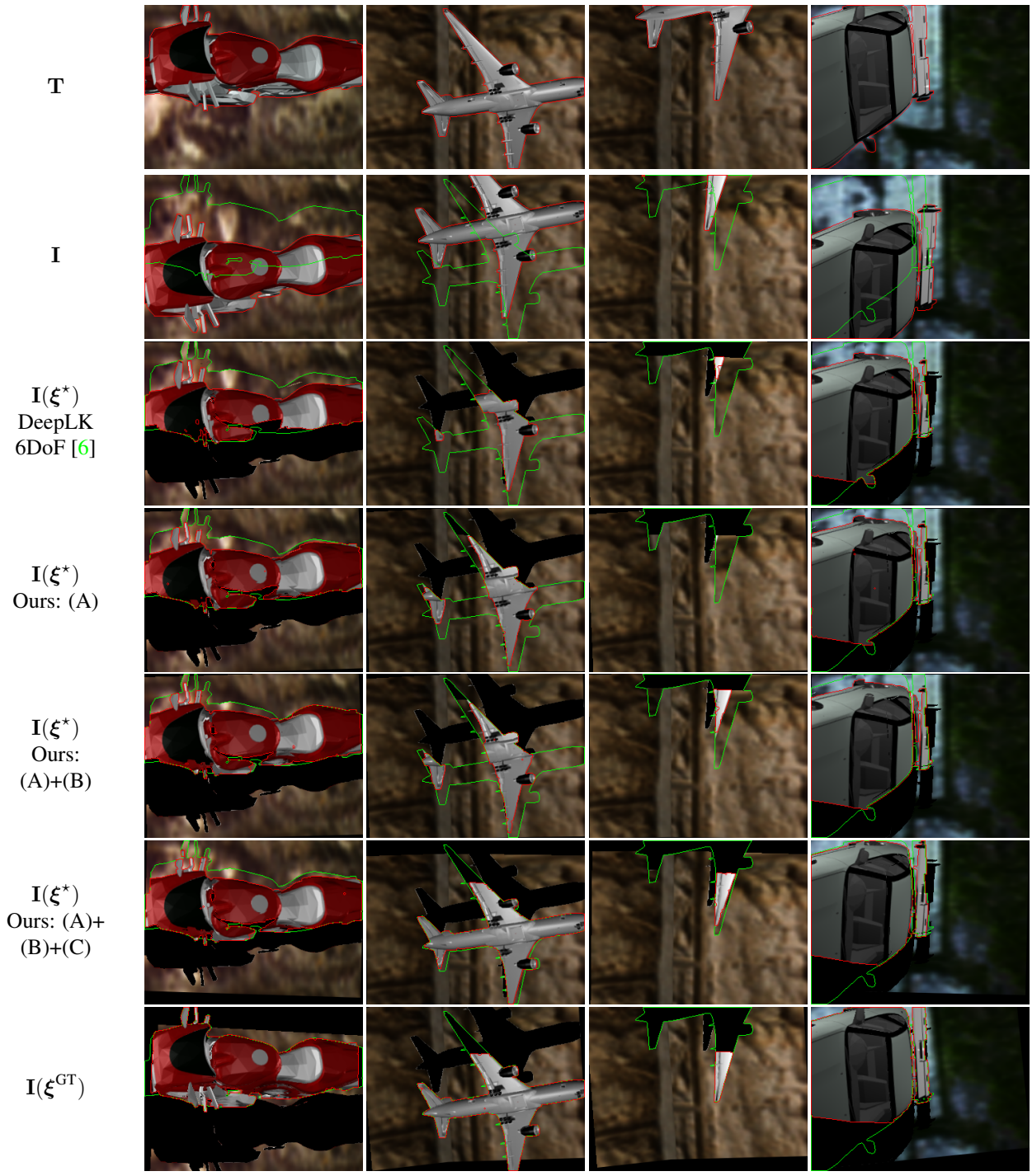

Figure 8: **Qualitative Comparisons of Our Method to Baselines on MovingObjects3D.** We compared the object motion $\boldsymbol{\xi}^\star$ estimated using our proposed modules (A)+(B)+(C) $\boldsymbol{\xi}^\star$ (row 6) to the optimal poses output from DeepLK 6DoF (row 3), ours (A) (row 4) and ours (A)+(B) (row 5) on the MovingObjects3D *validation* and *test* set. We visualize the warped image $\mathbf{I}(\boldsymbol{\xi})$ using the ground truth object motion $\boldsymbol{\xi}^{\text{GT}}$ (last row). In $\mathbf{I}$, we plot the instance boundary in red and that of $\mathbf{T}$ in green for qualitative comparison of the two shapes in 2D.

References

[1] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *Intl. J. of Computer Vision*, 56(3):221–255, Feb. 2004. 1

[2] J.-L. Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization. Technical report, University of Malaga, Sept. 2010. 4

[3] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719–722. IEEE, 2011. 4, 5

[4] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2012. 4

[5] R. Szeliski. Image alignment and stitching: A tutorial. *Foundations and Trends in Computer Graphics and Vision*, 2(1):1–104, 2007. 1

[6] C. Wang, H. K. Galoogahi, C.-H. Lin, and S. Lucey. Deep-lk for efficient adaptive object tracking. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2018. 3, 5, 8