Supplementary Material

A Total Loss

The total loss is calculated by following equation:

$$\mathcal{L} = \mathcal{L}_{ADD(-S)}(\widehat{P}^c, \widehat{P}^c_{gt}) + \mathcal{L}_{ADD(-S)}(\widehat{P}^f, \widehat{P}^f_{gt}) \tag{1}$$

$\widehat{P}^f$ and $\widehat{P}^c$ denote the prediction results for coarse and fine refinements. $\widehat{P}^c_{gt}$ and $\widehat{P}^f_{gt}$ indicate the ground truths. $\mathcal{L}_{ADD(-S)}$ indicates the ADD(-S) scores.

B Evaluation on YCB-Video and LineMOD-Occlusion

We train our model using Adam optimizer[1] with a learning rate of 110^{-3}, decayed by 0.5 every 100 epochs. The encoder of U-Net[2] consists of VGG-16[3] initialized by weights trained by ImageNet[4]. We follow the same convention in data processing as the previous works[5][6] for real images and fused images. We ignore the synthetic images because of their transfer gaps. During training, for each image, we add random gaussian noise to the ground truth as the initial pose, the mean and standard deviation of which are (0,0,0,0,0,0) and (0.5, 0.5, 0.5, 0.05, 0.05, 0.05). The values in parentheses are the mean or standard deviation of r, p, y, x, y, z, the units of which are radians and meters. Before iteration control learning, we set 5 steps for coarse refinements and 5 steps for fine refinements. For iteration control learning, we use ε-Greedy[10] to explore and exploit the actions and ε is set to 0.08. A example of reward figure is shown in Figure B1. Moreover, we consider to use TensorRT[8] to accelerate the forward propagation. TensorRT is provided as the software development kit for the NVIDIA hardware platform, including optimizer and runtime that delivers high and fast throughput for inference.

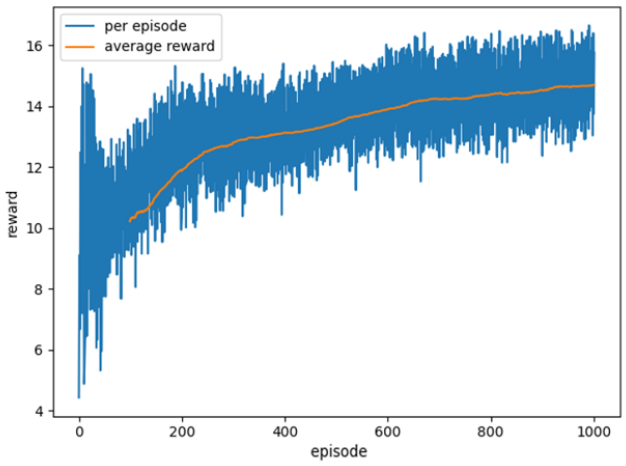

Fig. B1. Reward curve for DQN learning

Table B1 shows the result on the YCB-Video dataset[9]. We use PoseCNN[9] as the pose initializer, and we also follow the settings of RePOSE[5]. As shown in this table, our algorithm with DQN completely surpasses RePOSE in all the metrics. Moreover, for our algorithm without iteration control learning, it also achieves close results compared with RNNPose[6].

Table B1. The comparison of estimation accuracy with object pose refinement methods (DeepIM[7], RePOSE[5] and RNNPose[6]) on YCB-Video dataset. Bold value is the best result for each line. The number in parentheses is the average speed after TensorRT acceleration[8]. The underline value represents the better results of our algorithm with DQN compared with RePOSE.

Metric	PoseCNN[9]	DeepIM[7]	RePose[5]	RNNPose[6]	Ours w/o DQN	Ours with DQN
AUC, ADD(-S)	61.3	81.9	80.8	**83.1**	82.9	<u>81.0</u>
ADD(-S)	21.3	53.6	60.3	**66.4**	**66.4**	<u>63.6</u>
Speed	-	4.7	77.0	1.3	7.1(**79.0**)	33.1(**<u>79.0</u>**)

Because most of current state-of-art object pose estimation works choose LineMOD-Occlusion[11] dataset to do the evaluation with physically-based rendering[12], we add comparison with these works shown in table B2. During the experiment, the pose initializers of all the pose refinement algorithms(RePose[5], RNNPose[6] and ours) are set to GDRNet[13]. From the table B2, we find result of ZebraPose is the best, but it takes large runtime cost. Our algorithm without DQN is slightly weaker than ZebraPose, but the runtime cost is much better (4.1 $\nearrow$ 8.0(19.8)). Compared to other pose refinement algorithms, our algorithm is more balanced between accuracy and speed. With learned DQN, our algorithm still surpasses other real-time algorithms(CRT-6D[14](66.3 $\nearrow$ 68.4) and RePose[5](67.2 $\nearrow$ 68.4)).

Table B2. The comparison of estimation accuracy on LineMOD-Occlusion dataset. Bold value is the best result for each line. The number in parentheses is the average speed after TensorRT acceleration[8].

Methods	GDRNet [13]	CRT-6D [14]	ZebraPose [15]	RePose [5]	RNNPose [6]	Ours w/o DQN	Ours with DQN
Ape	46.8	53.4	**55.2**	48.1	54.8	54.7	51.7
Can	90.8	92.0	94.9	92.1	**95.8**	95.1	91.2
Cat	40.5	42.0	**56.6**	45.7	55.1	52.7	47.6
Driller	82.6	81.4	**94.7**	85.3	93.8	92.4	86.8
Duck	46.9	44.9	**60.9**	51.4	59.8	59.5	54.1
Eggbox	54.2	62.7	64.7	**66.2**	65.1	63.7	57.9
Glue	75.8	80.2	**84.5**	81.4	83.7	84.0	79.3
Holep.	60.1	74.3	**83.2**	67.1	83.1	82.8	78.4
Average	62.2	66.3	**74.3**	67.2	73.9	73.1	68.4
Speed(FPS)	40.7	25.1	4.1	**87.0**	1.3	8.0(19.8)	36.4(85.0)

C Adaptive cropping

To evaluate the effect of adaptive cropping on accuracy and speed, we compare it with different cropped sizes in Linemod dataset. We take an example

without DQN, as shown in Figure C1. From the figure, we can find with the decrease of cropped size, the speed improves but the accuracy will be reduced. Especially when cropped size is smaller than 128, the accuracy drops quickly. It is worth noting that the result with original shape (256×256) is not the best accuracy among the results of all the different cropped sizes. That is mainly because the larger of the input size, more noise will be brought to do the alignment. Our proposed adaptive cropping can achieve the best accuracy, even higher than the best results of all the different cropped sizes. Moreover, the speed can also reach the similar performance with that using the cropped size of 96.

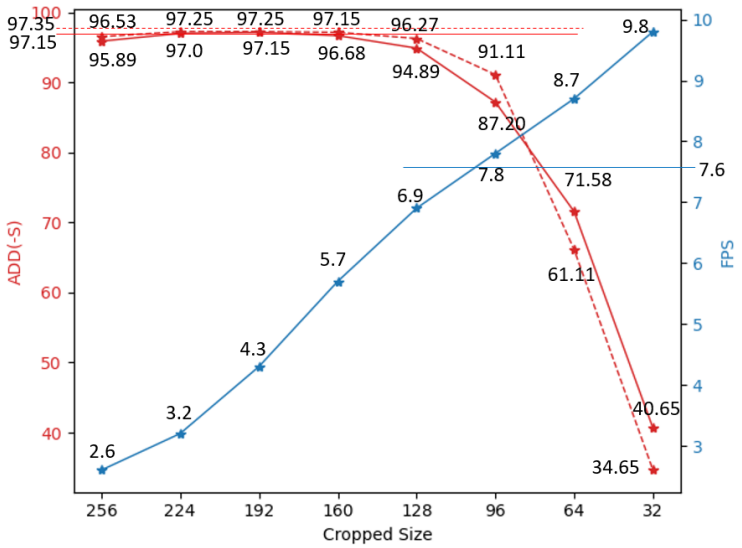

Fig. C1. Performance of different cropped sizes. The red line denotes the accuracy and the blue line denotes the runtime speed. The dashed and solid lines represent the result of poses initialized by PoseCNN and PVNET separately. 97.35 and 97.15 are the predicted accuracy, shown in the Table 1 of main text. 7.6 is the runtime speed(FPS).

D Qualitative Results

Some qualitative examples are shown as following(Figure D1 and Figure D2). C, F, and S denote coarse state, fine stage, and stop iteration separately. The first iteration is set to coarse stage by default. From the figure, we can find our learned DQN can be adaptive to the different initial errors to choose the suitable steps and stages for iteration, while using fixed number of iterations cannot. There is almost no redundant step for the entire iteration process using the learned DQN result.

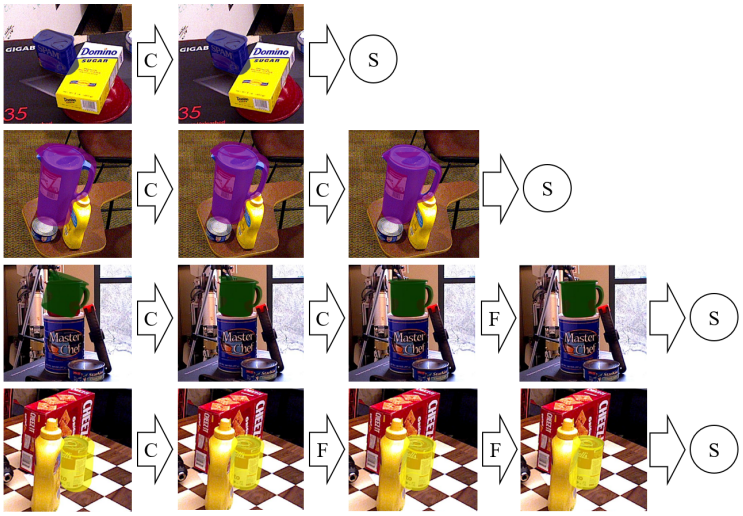

Fig. D1. Examples for YCB-Video dataset

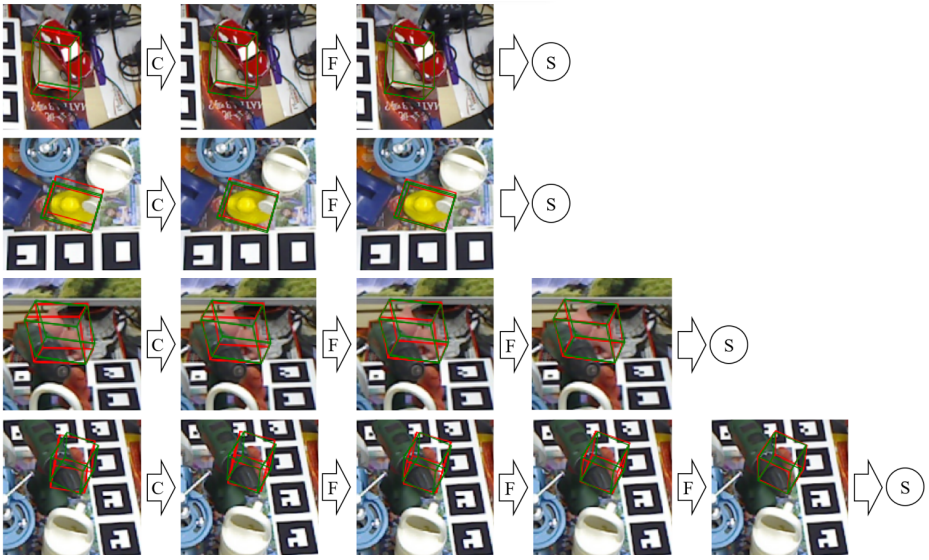

Fig. D2. Examples for Linemod-Occlusion dataset

References

1. Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In ICLR, 2015.

2. Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.
3. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
4. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).
5. S. Iwase, X. Liu, R. Khirodkar, R. Yokota, and K. M. Kitani, "Repose: Fast 6d object pose refinement via deep texture rendering," in Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 3303-3312.
6. Y. Xu, K.-Y. Lin, G. Zhang, X. Wang, and H. Li, "RNNPose: Recurrent 6-DoF Object Pose Refinement with Robust Correspondence Field Estimation and Pose Optimization," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 14880-14890.
7. Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "Deepim: Deep iterative matching for 6d pose estimation," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 683-698.
8. NVIDIA TensorRT, Mar. 2021, [online] Available: https://developer.nvidia.com/tensorrt/.
9. Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In Hadas KressGazit, Siddhartha S. Srinivasa, Tom Howard, and Nikolay Atanasov, editors, Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018, 2018.
10. V. Mnih et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529-533, 2015.
11. Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In European conference on computer vision, pages 536–551. Springer, 2014.
12. Hodaň, T., Sundermeyer, M., Drost, B., Labbé, Y., Brachmann, E., Michel, F., ... & Matas, J. (2020, August). BOP challenge 2020 on 6D object localization. In European Conference on Computer Vision (pp. 577-594). Springer, Cham.
13. G. Wang, F. Manhardt, F. Tombari, and X. Ji, "Gdr-net: Geometry-guided direct regression network for monocular 6d object pose estimation," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 16611-16621.
14. Castro, P., & Kim, T. K. (2023). CRT-6D: Fast 6D Object Pose Estimation with Cascaded Refinement Transformers. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (pp. 5746-5755).
15. Y. Su et al., "ZebraPose: Coarse to Fine Surface Encoding for 6DoF Object Pose Estimation," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 6738-6748.