RePOSE: Fast 6D Object Pose Refinement via Deep Texture Rendering

Shun Iwase[1] Xingyu Liu[1] Rawal Khirodkar[1] Rio Yokota[2] Kris M. Kitani[1]
[1]Carnegie Mellon University [2]Tokyo Institute of Technology

Abstract

We present RePOSE, a fast iterative refinement method for 6D object pose estimation. Prior methods perform refinement by feeding zoomed-in input and rendered RGB images into a CNN and directly regressing an update of a refined pose. Their runtime is slow due to the computational cost of CNN, which is especially prominent in multiple-object pose refinement. To overcome this problem, RePOSE leverages image rendering for fast feature extraction using a 3D model with a learnable texture. We call this deep texture rendering, which uses a shallow multi-layer perceptron to directly regress a view-invariant image representation of an object. Furthermore, we utilize differentiable Levenberg-Marquardt (LM) optimization to refine a pose fast and accurately by minimizing the distance between the input and rendered image representations without the need of zooming in. These image representations are trained such that differentiable LM optimization converges within few iterations. Consequently, RePOSE runs at 92 *FPS and achieves state-of-the-art accuracy of* 51.6% *on the Occlusion LineMOD dataset - a* 4.1% *absolute improvement over the prior art, and comparable result on the YCB-Video dataset with a much faster runtime. The code is available at https://github.com/sh8/repose.*

1. Introduction

In many applications of 6D object pose estimation like robotic grasping and augmented reality (AR), fast runtime is critical. State-of-the-art 6D object pose estimation methods [19, 40, 28] demonstrate that iterative 6D object pose refinement improves the accuracy largely. Nevertheless, since recent 6D object pose refinement methods [21, 19] directly regress an update of a pose to align a zoomed-in input image of an object against a template image (*e.g*., 3D rendering of that object) using a Convolutional Neural Network (CNN), we presume that the CNN's computational cost of zoomed-in inputs can be a bottleneck toward the real-time 6D object pose estimation.

We have mainly two choices of refinement strategies. As described, the former one is CNN-based direct regression, which generally requires large computational cost. The latter one is a classical non-linear optimization [24] which iteratively updates a pose by minimizing the photometric error between input and template images. Their runtime per iteration is quite fast. Since the photometric error explicitly considers each pixel, they can obtain enough details for accurate optimization without the need of zooming in. However, they can fail under diverse illumination or gross pose differences. Although non-linear least squares methods such as inverse compositional image alignment [5, 22] or active appearance models [13, 23] are extremely efficient, straightforward implementations of such methods can be unstable under significant illumination or pose changes. In addition, their runtime can be slower if many iterations are performed until convergence.

We leverage and improve the latter method to realize both quick and accurate refinement. In this paper, we propose RePOSE, a new feature-based non-linear optimization framework for 6D object pose refinement. The main tech-

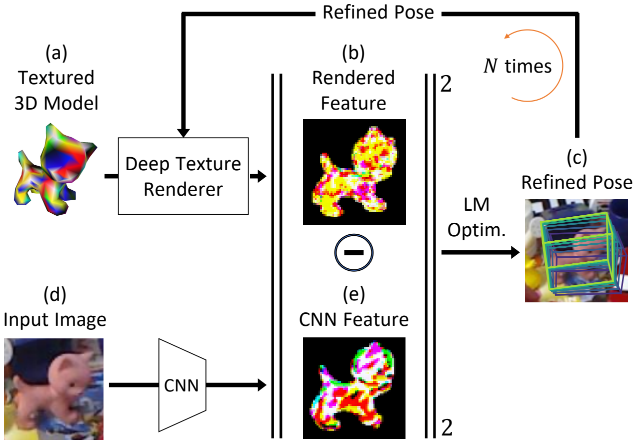


Figure 1: **RePOSE Framework:** (a) 3D model with deep texture is projected to obtain (b) the rendered image representation with the deep texture renderer. (c) The pose is refined iteratively by minimizing the projection error of the rendered image representation and (e) the CNN feature extracted from (d) the input image via Levenberg-Marquardt (LM) optimization.

nical insight presented in this work is that one can learn an image feature representation which is both robust for alignment and fast to compute. As stated earlier, the main impediment of CNN-based refinement methods is that the deep feature must be extracted during the refinement process iteratively. To remove this, we show that it is possible to directly render deep features using a simple graphics render. The rendering process decouples the shape of the object from the texture. At the time of rendering, texture is mapped to the 3D shape and then projected as a 2D image. Instead of mapping an RGB valued texture to the object, we can alternatively render a deep feature texture. Then, the rendered object can be directly aligned to the deep features of the input image. By retaining the deep feature representation during rendering, the pose alignment is robust and the refinement process becomes very efficient.

RePOSE refines an object pose by minimizing the distance between the deep features of the input and rendered images. Since the input image is fixed during iterative refinement, its feature is only computed once using a CNN. In contrast, the deep feature of the template image are directly generated using a simple computer graphics renderer. The rendering process takes less than a millisecond which greatly increases the speed of the iterative refinement process. The deep feature representation is learned such that nonlinear optimization can be easily performed through a differentiable LM optimization network [24]. We experimentally found 5 iterations are enough to converge, which contributes to fast 6D object pose refinement.

RePOSE has several practical advantages over recent CNN-based regression methods: 1) RePOSE can be exceptionally fast. — In the case of 1 iteration, RePOSE runs at 181 FPS for 5 objects and 244 FPS for 1 object, 2) RePOSE is data efficient. — Since RePOSE considers projective geometry explicitly, there is no need to learn the mapping of the deep feature into an object's pose from training data. In our experiments, we show that RePOSE achieves better or comparable performance with much fewer number of training images than prior methods, and 3) RePOSE does not request RGB textures of a 3D model. — It has been known that RGB texture scanning has troubles with metalic, dark-colored, or transparent objects even with the latest 3D scanner [1]. We believe that the requirement of RGB textures by recent CNN-based regression methods [21, 19] makes the implementation in the real world more challenging.

We evaluate RePOSE on three popular 6D object estimation datasets - LineMOD [15], the challenging Occlusion LineMOD [6], and YCB-Video [39]. RePOSE sets a new state of the art on the Occlusion LineMOD (51.6%) [6] dataset and achieves comparable performance on the other datasets with much faster speed (80 to 92 FPS with 5 iterations). Additionally, we perform ablations to validate the effectiveness of our proposed methods.

2. Related Work

Two-stage pose estimation methods Recently, Oberweger [26], PVNet [27], DPOD [40], and HybridPose [33] have shown excellent performance on 6D object pose estimation using a two-stage pipeline to estimate a pose: (i) estimating a 2D representation (*e.g*. keypoints, dense correspondences, edge vectors, symmetry correspondences), (ii) PnP algorithm [20, 11] for pose estimation. DOPE [36] and BB8 [28] estimate the corners of the 3D bounding box and run a PnP algorithm. Instead of regarding the corners as keypoints, PVNet [27] places the keypoints on the object surface via the farthest point sampling algorithm. PVNet also shows that their proposed voting-based keypoint detection algorithm is effective especially for occluded objects. HybridPose [33] uses multiple 2D representations including keypoints, edge vectors, and symmetry correspondences and demonstrates superior performance through constraint optimization. DPOD [40] takes advantage of the dense correspondences using a UV map as a 2D representation. However, since the PnP algorithm is sensitive to small errors in the 2D representation, it is still challenging to estimate the object pose especially under occlusion. RePOSE adopts PVNet [27] as the initial pose estimator using the official implementation.

Pose refinement networks Recent works [39, 34, 40, 33, 21] have demonstrated that using a pose refinement network after the initial pose estimator is effective for 6D object pose estimation. For practical applications, the runtime of the pose refinement network is crucial. PoseCNN [39] and AAE [34] incorporates an ICP algorithm [41] using depth information to refine the pose with a runtime of around 200 ms. SSD6D [17] and HybridPose [33] proposed to refine the pose by optimizing a modification of reprojection error. DeepIM [21], DPOD [40], and CosyPose [19] introduce a CNN-based refinement regression network using the zoomed-in input image and a rendered object image. Their methods require a high-quality texture map of a 3D model to compare the images. However, it is still challenging to obtain accurate texture scans of metalic, dark-colored, or transparent objects. NeMO [38] proposes a pose refinement method using the standard differentiable rendering and learning the texture of a 3D model via contrastive loss. However, gradient descent is used for optimization, hence, it takes more than 8s for inference and is not fast enough for real-time applications.

Non-linear least squares optimization Non-linear least squares optimization is widely used in machine learning. In computer vision, it is often utilized to find an optimal pose which minimizes the reprojection error or photometric error [4, 25, 32]. Recently some works [35, 37, 12] incorporate

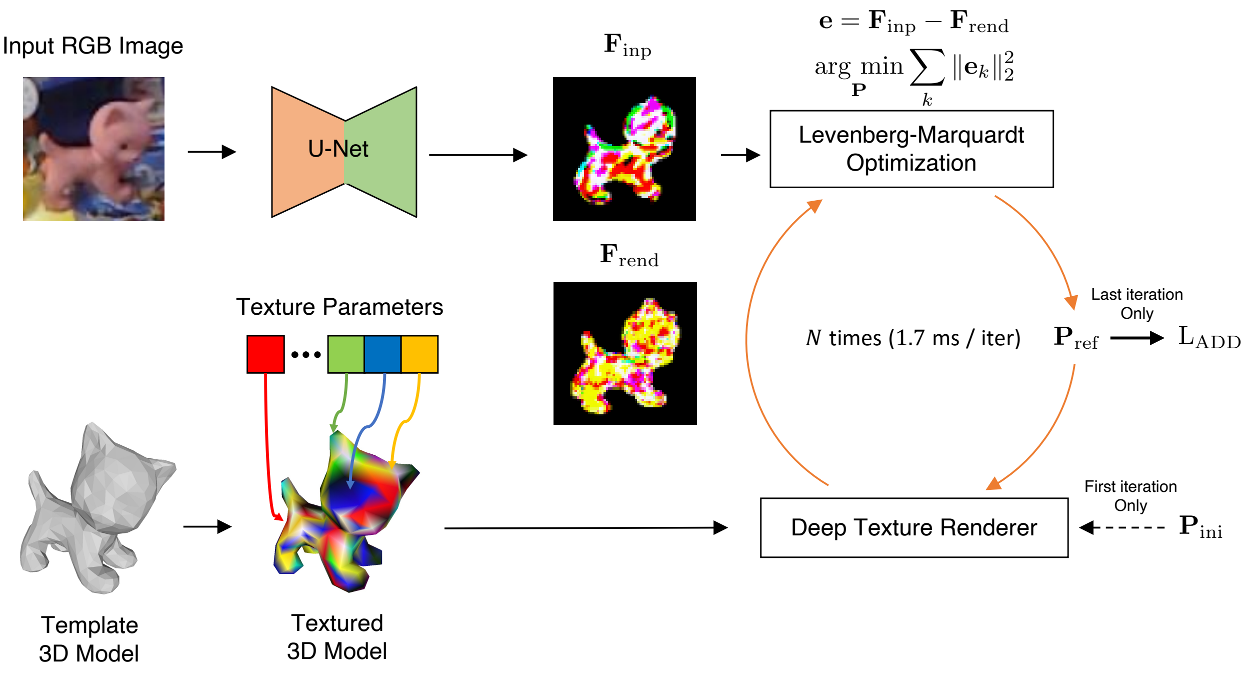

Input RGB Image
U-Net
$\mathbf{F}_{\text{inp}}$
$\mathbf{e} = \mathbf{F}_{\text{inp}} - \mathbf{F}_{\text{rend}}$
$\arg\min_{\mathbf{P}} \sum_k \|\mathbf{e}_k\|_2^2$
Levenberg-Marquardt Optimization
$\mathbf{F}_{\text{rend}}$
Texture Parameters
N times (1.7 ms / iter)
Last iteration Only
$\mathbf{P}_{\text{ref}}$
L_{ADD}
First iteration Only
$\mathbf{P}_{\text{ini}}$
Deep Texture Renderer
Template 3D Model
Textured 3D Model

Figure 2: Overview of the RePOSE refinement network. Given an input image $\mathbf{I}$ and the template 3D model M with deep textures, U-Net and deep texture renderer output features $\mathbf{F}_{\text{inp}}$ and $\mathbf{F}_{\text{rend}}$ respectively. We use Levenberg-Marquardt optimization [24] to obtain the refined pose $\mathbf{P}_{ref}$. The refined pose $\mathbf{P}_{ref}$ after N iterations is used to compute the loss $L_{ADD(-S)}$. The pre-trained encoder of the initial pose estimator is used. The decoder of U-Net and deep textures (seed parameters, and fc layers) are trained to minimize $L_{\text{ADD(-S)}}$ and L_{diff}.

non-linear least squares algorithms like Gauss-Newton and Levenberg-Marquardt [24] into a deep learning network for efficient feature optimization in VisualSLAM. RePOSE is inspired by similar formulation as in [35].

3. RePOSE: Fast 6D Object Pose Refinement

Given an input image $\mathbf{I}$ with a ground-truth object pose $\mathbf{P}_{\text{gt}}$ and the template 3D model $\mathcal{M}$, RePOSE predicts pose $\hat{\mathbf{P}}$ of model M which matches $\mathbf{P}_{\text{gt}}$ in $\mathbf{I}$. We extract a feature $\mathbf{F}_{\text{inp}}$ from image $\mathbf{I}$ using a CNN Φ *i.e.* $\mathbf{F}_{\text{inp}} = \Phi(\mathbf{I})$. RePOSE then refines the initial pose estimate $\mathbf{P}_{\text{ini}} = \Omega(\mathbf{I})$ where Ω is any pose estimation method like PVNet [27] and PoseCNN [39] in real time using differentiable Levenberg–Marquardt (LM) optimization [24]. RePOSE renders the template 3D model with learnable deep textures in pose $\mathbf{P}$ to extract feature $\mathbf{F}_{\text{rend}}$. The pose refinement is performed by minimizing the distance between $\mathbf{F}_{\text{inp}}$ and $\mathbf{F}_{\text{rend}}$. We now describe in detail (1) $\mathbf{F}_{\text{inp}}$ extraction, (2) $\mathbf{F}_{\text{rend}}$ extraction and finally (3) the pose refinement using LM optimization.

3.1. Feature Extraction of an Input Image $\mathbf{F}_{\text{inp}}$

We adopt a U-Net [29] architecture for the CNN Φ. The decoder outputs a deep feature map for every pixel in $\mathbf{I}$. The per-pixel feature $\mathbf{F}_{\text{inp}} \in \mathbb{R}^{w \times h \times d}$ is extracted by the decoder. Figure 1 (b) provides a visual illustration of $\mathbf{F}_{\text{inp}}$ extracted from the input image $\mathbf{I}$. Note that the channel depth d is a flexible parameter but we found $d = 3$ to be optimal. The pre-trained weights of PVNet [27] or PoseCNN [39] are used for the encoder and only the decoder is trained while training RePOSE.

3.2. Template 3D Model Rendering $\mathbf{F}_{\text{rend}}$

The template 3D model $\mathcal{M}$ with pose $\mathbf{P} = \{\mathbf{R}, \mathbf{t}\}$ where $\mathbf{R}$ is 3D rotation and $\mathbf{t}$ is 3D translation, is projected to 2D to render the feature $\mathbf{F}_{\text{rend}}$. Let the template 3D model $\mathcal{M} = \{\mathcal{V}, \mathcal{C}, \mathcal{F}\}$ be represented by a triangular watertight mesh consisting of N vertices $\mathcal{V} = \{V_n\}_{n=1}^{N}$ where $V_n \in \mathbb{R}^3$, faces $\mathcal{F}$ and deep textures $\mathcal{C}$. V_n is the 3D coordinate of the vertex in the coordinate system centered on the object. Each vertex V_n has a corresponding vertex learnable texture $\mathbf{C}_n \in \mathbb{R}^d$, $\mathcal{C} = \{\mathbf{C}_n\}_{n=1}^{N}$, which is learned. Note that the dimensions of the vertex learnable texture d must match depth dimension of input image feature $\mathbf{F}_{\text{inp}}$ so that they can be compared during alignment.

RePOSE projects the 3D mesh onto to the image plane using a pinhole camera projection function π (homogeneous to inhomogeneous coordinate conversion). Specifically, we

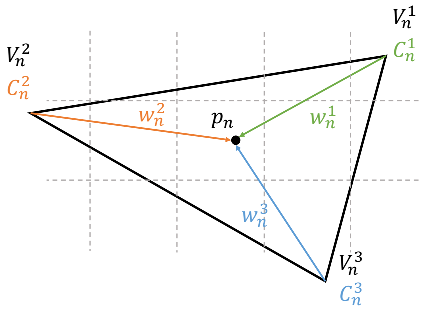


Figure 3: Rasterization of deep textures into a pixel p_n as the weighted sum of $\mathbf{C}_n^i$ using w_n^i as weights in the barycentric coordinate system, $\sum_i^3 w_n^i = 1$.

map the vertex $\mathbf{V}_n$ to $\mathbf{v} \in \mathbb{R}^2$ using eq 1.

$$\mathbf{v}_n = \pi \left(\mathbf{V}_n \mathbf{R}^\top + \mathbf{t}^\top \right) \ \ \forall \, n \tag{1}$$

The vertex deep textures $\mathbf{C}_n \in \mathbb{R}^3$ are learnable and computed using a 2-layer fully-connected network. The deep texture at each pixel is calculated by rasterization using the deep textures $\mathbf{C}_n$ in barycentric coordinates w as shown in Figure 3. This operation can be parallelized using a GPU. Our custom implementation of the [16]'s renderer takes less than 1 ms to render $\mathbf{F}_{\text{rend}}$. $\mathbf{F}_{\text{rend}}(x, y)$ at a pixel location (x, y) is computed as follows:

$$\mathbf{F}_{\text{rend}}(x, y) = \sum_{i=1}^{3} w_n^i C_n^i \tag{2}$$

where the triangular face index n corresponding to the pixel p_n at (x, y) is found by ray tracing and w_i is the normalized barycentric weight corresponding to the coordinates (x, y) inside the triangle (Figure 3). Simply put, the rendered deep feature $\mathbf{F}_{\text{rend}}(x, y)$ is a linear combination of deep textures of the three projected vertices.

$\mathbf{F}_{\text{rend}}$ is end-to-end learnable by backpropagation. The gradient of $\mathbf{F}_{\text{rend}}$ with respect to the three deep textures of the triangle $\{C_n^i\}_{i=1}^3$ is as follows:

$$\frac{\partial \mathbf{F}_{\text{rend}}(x, y)}{\partial C_n^i} = w_n^i. \tag{3}$$

Note that $\mathbf{F}_{\text{rend}}$ is the output of a non-linear function Ψ of the template 3D model $\mathcal{M}$ and its pose $\mathbf{P}$, *i.e.*, $\mathbf{F}_{\text{rend}} = \Psi(\mathbf{P}, \mathcal{M})$ where Ψ is the deep texture renderer (Figure 2).

3.3. Levenberg-Marquardt (LM) Optimization

After computing $\mathbf{F}_{\text{inp}}$ (Section 3.1) and $\mathbf{F}_{\text{rend}}$ (Section 3.2), the optimal pose $\hat{\mathbf{P}}$ is calculated by minimizing the following objective function:

$$\mathbf{e} = \text{vec}(\mathbf{F}_{\text{inp}}) - \text{vec}(\mathbf{F}_{\text{rend}}), \tag{4}$$

$$\hat{\mathbf{P}} = \arg\min_{\mathbf{P}} \sum_k ||e_k||_2^2, \tag{5}$$

where e_k denotes the k^{th} element of the error $\mathbf{e} \in \mathbb{R}^{whd}$ and is the element-wise difference between the flattened values of $\mathbf{F}_{\text{inp}}$ and $\mathbf{F}_{\text{rend}}$. To perform optimization efficiently, we only use the error $\mathbf{e}$ in the pixel where the mask of $\mathbf{F}_{\text{rend}}$ exists.

We solve this non-linear least squares problem using the iterative Levenberg-Marquardt (LM) algorithm. The update rule for the pose $\mathbf{P}$ is as follows:

$$\Delta \mathbf{P} = \left(\mathbf{J}^T \left(\mathbf{e} \right) \mathbf{J} + \lambda \mathbf{I} \right)^{-1} \mathbf{J}^T \left(\mathbf{e} \right) \mathbf{e}, \tag{6}$$

$$\mathbf{P}_{i+1} = \mathbf{P}_i + \Delta \mathbf{P}, \tag{7}$$

where $\mathbf{J}$ is the Jacobian of the objective with respect to the pose $\mathbf{P}$, and λ is a learnable step size.

The Jacobian $\mathbf{J}$ can be decomposed as:

$$\mathbf{J} = \frac{\partial \mathbf{F}_{\text{rend}}}{\partial \mathbf{P}} = \frac{\partial \mathbf{F}_{\text{rend}}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \mathbf{P}} \tag{8}$$

where $\mathbf{x}$ is a vector of all 2D image coordinate. We compute $\frac{\partial \mathbf{F}_{\text{rend}}}{\partial \mathbf{x}}$ using a finite difference approximation and $\frac{\partial \mathbf{x}}{\partial \mathbf{P}}$ is computed analytically. Please refer supplemental for the details.

We minimize a loss function $\mathcal{L}_{\text{ADD(-S)}}$ based on the ADD(-S) score:

$$\mathcal{L}_{\text{ADD(-S)}} = S_{\text{ADD(-S)}}(\mathbf{P}, \mathbf{P}_{\text{gt}}) \tag{9}$$

where S is the function used to calculate the distance used in the ADD(-S) score. Additionally, we also minimize a loss function $\mathcal{L}_{\text{diff}}$ which ensures the value of the objective function is minimized when the pose $\mathbf{P}$ is equal to $\mathbf{P}_{\text{gt}}$:

$$\mathbf{d} = \text{vec}(\mathbf{F}_{\text{inp}}) - \text{vec}(\Psi(\mathbf{P}_{\text{gt}}, \mathcal{M})), \tag{10}$$

$$\mathcal{L}_{\text{diff}} = \sum_k ||d_k||_2^2. \tag{11}$$

The minimization of these two loss functions through LM optimization allows our refinement network to learn representations of the input image as well as the rendered object image, which helps in predicting the optimal pose.

$$\mathcal{L} = \mathcal{L}_{\text{ADD(-S)}} + \alpha \mathcal{L}_{\text{diff}} \tag{12}$$

where α is a hyperparameter.

We show the RePOSE framework in Algorithm 1. Note, all the operations inside the LM optimization (Equations (6) and (7)) are differentiable allowing us to learn deep textures $\mathcal{C}$ and Φ using backpropagation.

4. Experiments

4.1. Implementation Details

We train our model using Adam optimizer [18] with a learning rate of 1×10^{-3}, decayed by 0.5 every 100 epochs.

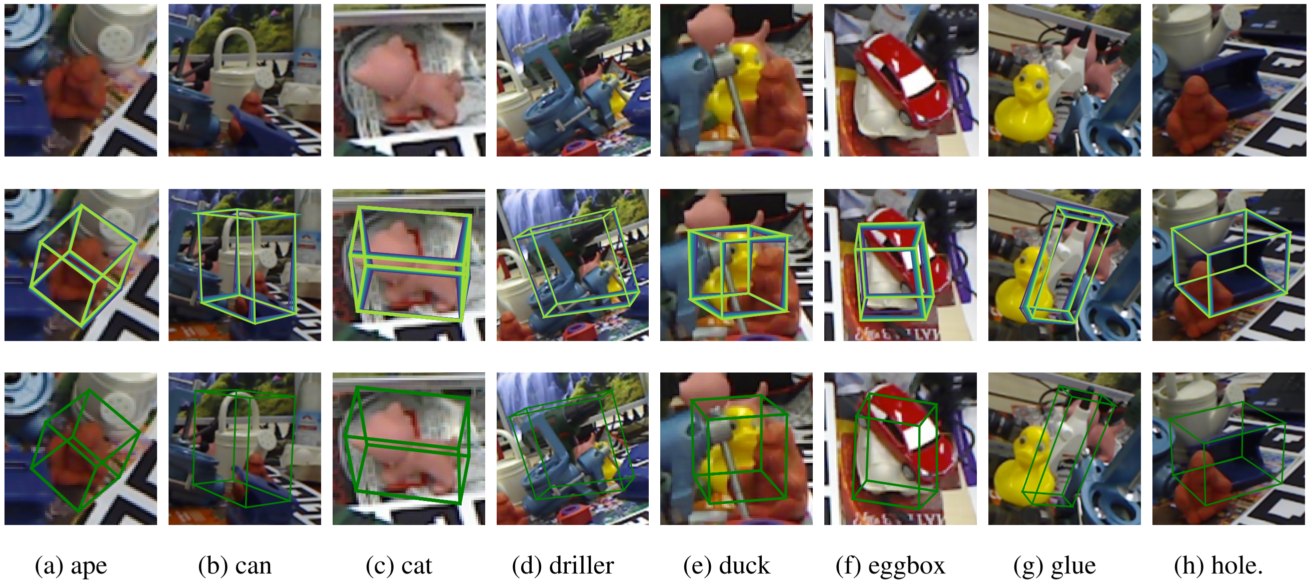

(a) ape (b) can (c) cat (d) driller (e) duck (f) eggbox (g) glue (h) hole.

Figure 4: Example results on the Occlusion LineMOD dataset [6]. We show an input RGB image, refined poses, and ground-truth pose from the top to bottom. The color of 3D bounding boxes are changed from purple to lightgreen as optimization progresses.

```
Algorithm 1: RePOSE Training
$\mathcal{V}$ = VERTICESOF3DMODEL();
$\mathcal{F}$ = FACESOF3DMODEL();
$\mathcal{C}$ = INITIALIZETEXTUREPARAMETERS();
# Iterate over Training Data
for $\mathbf{P}_{ini}, \mathbf{P}_{gt}, \mathbf{I}$ do
    $\mathbf{F}_{\text{inp}}$ = UNET ($\mathbf{I}$);
    $\mathbf{P} = \mathbf{P}_{ini}$;
    for $t$ times do
        $\mathbf{F}_{\text{rend}}$ = DEEPTEXTURERENDER ($\mathbf{P}, \mathcal{V}, \mathcal{F}, \mathcal{C}$);
        $\mathbf{e} = \text{vec}(\mathbf{F}_{\text{inp}}) - \text{vec}(\mathbf{F}_{\text{rend}})$;
        $\mathbf{J}$ = JACOBIAN ($\mathbf{F}_{\text{rend}}, \mathbf{P}, \mathcal{V}$);
        $\Delta\mathbf{P}$ = POSEUPDATE ($\mathbf{e}, \mathbf{J}, \mathbf{P}$);
        $\mathbf{P} = \mathbf{P} + \Delta\mathbf{P}$; # Update Pose
    $\mathbf{P}_{ref} = \mathbf{P}$;
    $\mathcal{L}$ = LOSS ($\mathbf{P}_{\text{ref}}, \mathbf{P}_{\text{gt}}, \mathbf{V}$);
    UPDATEPARAMETERS($\mathcal{L}, \mathcal{C}$, UNET);
```

The number of channels d in $\mathbf{F}_{\text{inp}}$ and $\mathbf{F}_{\text{rend}}$ is set to 3 using grid search, and iterations t in LM optimization is set to 5. We used pretrained PVNet [27] on the LineMOD and Occlusion LineMOD datasets, and PoseCNN [39] on the YCB-Video [39] dataset as the initial pose estimator Ω. The encoder of U-Net [29] consisting of ResNet-18 [14] shares its weights with the PVNet, and PoseCNN and only the weights of the decoder are trained. Therefore, RePOSE simply can reuse the deep features extracted from the initial pose estimator, which reduces the computational cost.

Following [27], we also add 500 synthetic and fused images for LineMOD and 20K synthetic images for YCB-Video to avoid overfitting during training. In accordance with the convention, to evaluate the scores on the Occlusion LineMOD dataset, we use the model trained by using only the LineMOD dataset.

4.2. Datasets

All experiments are performed on the LineMOD [15], Occlusion LineMOD [6], and YCB-Video [39] datasets. The LineMOD dataset contains images of small textureless objects in a cluttered scene under different illumination. High-quality template 3D models of the objects in the images are also provided for render and compare based pose estimation. The Occlusion LineMOD dataset is a subset of the LineMOD dataset focused mainly on the occluded objects. YCB-Video [39] dataset contains images of objects from the YCB-object set [10]. We use ADD (-S) [15] and AUC of ADD(-S) scores as our evaluation metrics.

4.3. Evaluation Metrics

ADD(-S) score. ADD(-S) score [15, 39] is a standard metric which calculates the average distance between objects transformed by the predicted pose $\hat{\mathbf{P}} = \{\hat{\mathbf{R}}, \hat{\mathbf{t}}\}$, and the ground-truth pose $\mathbf{P}_{\text{gt}} = \{\mathbf{R}_{\text{gt}}, \mathbf{t}_{\text{gt}}\}$ using vertices $\mathbf{V}_i$ of the template 3D model $\mathcal{M}$. The distance is calculated as

Table 1: Results on the YCB-Video dataset using *RGB only*. The results for DeepIM [21] are computed using the official pre-trained model, and the score inside the parentheses are the reported results from the paper. Refinement FPS denotes FPS of running only a pose refinement network. RePOSE w/ track includes the runtime for CNN feature extraction of a real image. FPS is reported with refinement of 5 objects.

Metric	PoseCNN [39]	DeepIM [21]		PVNet [27]	CosyPose [19]		RePOSE			RePOSE w/ track		
AUC, ADD(-S)	61.3	74.0	75.5 (81.9)	73.4	84.1	**84.5**	70.5	79.4	80.8	70.1	80.6	82.0
AUC, ADD-S	75.2	83.1	83.1 (88.1)	-	**89.8**	**89.8**	80.4	85.9	86.7	79.9	87.2	88.5
ADD(-S)	21.3	43.2	53.6	-	74.3	**75.6**	41.7	58.9	60.3	40.2	61.6	62.1
Refinement FPS	-	22	6	-	26	13	**181**	111	80	125	90	71
#Iterations	-	1	4	-	1	2	1	3	5	1	3	5

Table 2: Comparison of RePOSE on Linemod dataset with recent methods including PVNet [27], DPOD [40], HybridPose [33], and EfficientPose [9] using the ADD(-S) score. # of wins denotes in how many objects the method achieves the best score.

Object	PVNet	DPOD	HybridPose	EfficientPose	RePOSE
Ape	43.6	87.7	63.1	**89.4**	79.5
Benchvise	99.9	98.5	99.9	99.7	**100**
Camera	86.9	96.1	90.4	98.5	**99.2**
Can	95.5	99.7	98.5	99.7	**99.8**
Cat	79.3	94.7	89.4	96.2	**97.9**
Driller	96.4	98.8	98.5	**99.5**	99.0
Duck	52.6	86.3	65.0	**89.2**	80.3
Eggbox	99.2	99.9	**100**	**100**	**100**
Glue	95.7	98.7	98.8	**100**	98.3
Holepuncher	81.9	86.9	89.7	95.7	**96.9**
Iron	98.9	**100**	**100**	99.1	**100**
Lamp	99.3	96.8	99.5	**100**	99.8
Phone	92.4	94.7	94.9	98.5	**98.9**
Average	86.3	95.2	91.3	**97.4**	96.1
# of wins	0	1	2	6	**8**

Table 3: Comparison of RePOSE on Occlusion LineMOD dataset with recent methods including PVNet [27], DPOD [40], and HybridPose [33] using the ADD(-S) score. Note, we exclude EfficientPose [9] as it is trained on the Occlusion LineMOD dataset. # of wins denotes in how many objects the method achieves the best score.

Object	PVNet	DPOD	HybridPose	RePOSE
Ape	15.8	-	20.9	**31.1**
Can	63.3	-	75.3	**80.0**
Cat	16.7	-	24.9	**25.6**
Driller	65.7	-	70.2	**73.1**
Duck	25.2	-	27.9	**43.0**
Eggbox	50.2	-	**52.4**	51.7
Glue	49.6	-	53.8	**54.3**
Holepuncher	39.7	-	**54.2**	53.6
Average	40.8	47.3	47.5	**51.6**
# of wins	0	-	2	**6**

follows;

$$\frac{1}{N}\sum_{i}^{N}||\left(\hat{\mathbf{R}}\mathbf{V}_i+\hat{\mathbf{t}}\right)-(\mathbf{R}_{gt}\mathbf{V}_i+\mathbf{t}_{gt})|| \tag{13}$$

For symmetric objects such as `eggbox` and `glue`, we use the following distance metric,

$$\frac{1}{N}\sum_{i}^{N}\min_{0\leq j\leq N}||\left(\hat{\mathbf{R}}\mathbf{V}_i+\hat{\mathbf{t}}\right)-(\mathbf{R}_{gt}\mathbf{V}_j+\mathbf{t}_{gt})|| \tag{14}$$

The predicted pose is considered correct if this distance is smaller than 10% of the target object's diameter. AUC of ADD(-S) computes the area under the curve of the distance used in ADD(-S). The pose predictions with distance larger than 0.1m are not included in computing the AUC. We use AUC of ADD(-S) to evaluate the performance on the YCB-Video dataset [39].

4.4. Quantitative Evaluations

Results on the LineMOD and Occlusion LineMOD datasets. As shown in Tables 2 and 3, RePOSE achieves the state of the art ADD(-S) scores on the Occlusion LineMOD dataset. In comparison to PVNet [27], RePOSE successfully refines the initial pose estimate in all the objects, achieving an improvement of 9.8% and 10.8% on the LineMOD and Occlusion LineMOD dataset respectively. On the LineMOD dataset, our score is comparable to the state-of-the art EfficientPose [9]. The key difference is mainly on `ape` and `duck` where our initial pose estimator PVNet [27] performs poorly. Interestingly, for small objects like `ape` and `duck` in the Occlusion LineMOD dataset, we show a significant improvement of 10.2 and 15.1 respectively over the prior art HybridPose [33].

Results on the YCB-Video dataset. Table 1 shows the result on the YCB-Video dataset [39]. We also performed experiments using RePOSE as a 6D object tracker using the tracking algorithm proposed in [21]. RePOSE achieves comparable performance with other methods with a 4 times faster runtime of 80 FPS for refinement of 5 objects. Further, the result with tracking demonstrates that RePOSE is useful as a real-time 6D object tracker. Note, the scores are heavily affected by the use and amount of synthetic data and various data augmentation [19]. For instance, Cosy-

Table 4: Ablation study of feature representation, feature warping, and a refinement network on the LineMOD dataset. RGB denotes pose refinement using photometric error. FW denotes feature warping after extraction from a CNN or deep texture rendering following first iteration. DPOD denotes using DPOD's refinement network and PVNet as an initial pose estimator. FW, DPOD, and RePOSE are trained with the same dataset, we report the ADD(-S) scores.

Object	PVNet [27]	RGB	CNN w/ FW	DPOD	Ours w/ FW	Ours
Ape	43.6	5.81	65.4	51.2	75.9	**79.5**
Benchvise	99.9	75.6	99.8	99.5	**100**	**100**
Camera	86.9	7.06	96.3	91.1	98.2	**99.2**
Can	95.5	3.05	99.1	95.7	99.4	**99.8**
Cat	79.3	3.00	88.6	92.4	92.7	**97.9**
Driller	96.4	80.9	7.6	98.2	98.7	**99**
Duck	52.6	0.00	76.2	71.3	**84.6**	80.3
Eggbox	99.2	8.64	96.4	99.9	**100**	**100**
Glue	95.7	5.40	97.2	97.6	98.2	**98.3**
Holepuncher	81.9	18.7	77.2	89.7	95.1	**96.9**
Iron	98.9	40.7	98.7	97.9	99.7	**100**
Lamp	99.3	34.9	91.8	95.5	100	99.8
Phone	92.4	14.6	94.9	97.2	98.7	**98.9**
Average	86.3	23.0	90.7	90.5	95.5	**96.1**

Table 5: Ablation study of feature representation, feature warping, and a refinement network on the Occlusion LineMOD dataset. We report the ADD(-S) scores, all other details are same as in Table 4.

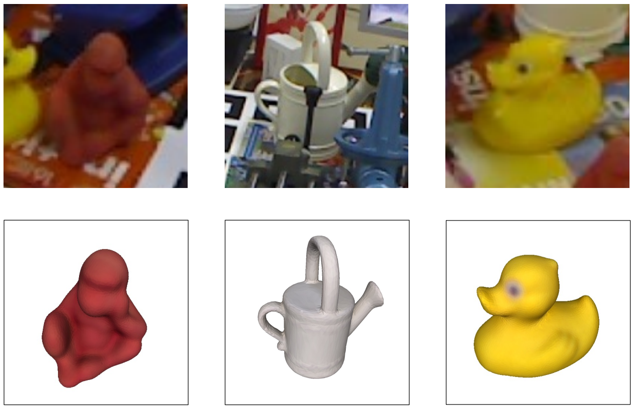
Object	PVNet [27]	RGB	CNN w/ FW	DPOD	Our w/ FW	Ours
Ape	15.8	4.96	22.7	22.0	25.8	**31.1**
Can	63.3	5.22	66.4	71.1	61.3	**80.0**
Cat	16.7	0.17	11.7	21.9	19.4	**25.6**
Driller	65.7	61.7	72.1	68.3	71.1	**73.1**
Duck	25.2	1.80	36.5	30.8	40.8	**43.0**
Eggbox	50.2	7.75	45.4	42.4	47.7	**51.7**
Glue	49.6	1.88	45.6	41.3	49.4	**54.3**
Holepuncher	39.7	21.5	40.8	43.3	40.2	**53.6**
Average	40.8	13.1	42.9	42.6	44.5	**51.6**

Pose [19] used one million synthetic images during training, making it hard to compare against fairly. However, our method achieves comparable performance using 500 times less training images.

4.5. Ablation Study

All ablations for RePOSE are conducted on the LineMOD and Occlusion LineMOD datasets using PVNet [27] as an initial pose estimator. We report the results in Tables 4 and 5.

RGB vs Deep Texture. Instead of using learnable deep textures $\mathcal{C}$, we perform experiments using an original RGB image and rendered image with scanned colors. The inference is all the same except we are using photometric error between two images. The experimental result reported in Tables 4 and 5 show that the ADD(-S) score drops significantly after optimization in all the objects using RGB representation. As illustrated in Figure 5, the LineMOD dataset has three main challenges which makes the pose refinement using the photometric error difficult — 1) Illumination changes between the input RGB image and synthetic rendering, 2) Poor image gradients due to texture-less objects, 3) Background confusion *i.e.* the background color is similar to the object's color. The ADD(-S) scores drop largely due to these key reasons. On the contrary, RePOSE with learnable deep textures is able to converge within few iterations because of the robustness of deep textures to the above challenges. Tables 4 and 5 clearly demonstrate the effectiveness of our learnable deep textures over using scanned colors for the template 3D model.



(a) Ape (b) Can (c) Duck

Figure 5: Comparison of object's appearance between an input RGB image and rendered image. Difference of illumination makes pose refinement in RGB space challenging. Furthermore, RGB images may have the region with the same color as the object. This background noise becomes an obstacle in terms of convergence properties. These texture-less objects make it challenging to compute the image gradient which is essential to optimize a pose.

CNN with Feature Warping vs Feature Rendering. Feature warping (FW) is commonly used to minimize photometric or feature-metric error through a non-linear least squares such as Gauss-Newton or Levenberg-Marquardt method [3, 2]. We conduct an experiment to compare a CNN with feature warping and our proposed feature rendering using the deep texture renderer. In a CNN with feature warping, $\mathbf{F}_{\text{rend}}$ is extracted in the same fashion as the $\mathbf{F}_{\text{inp}}$ using a CNN on a normalized synthetic rendering of the template 3D model. This is done just once, following which the feature is warped based on the updated pose at each iteration. The result is shown in Tables 4 and 5. On the LineMOD dataset, we observed on average small improvments by the feature warping. The ADD(-S) score only allows the pose estimator to have an mean vertex distance error of 10% of the object's diameter. In this task, this means only 2 to 3 pixel displacement error in 2D image space is allowed especially for small objects. However, it is challenging to train a CNN to extract features with accurate image gradients required for fine-grained pose refinement. On the contrary, our deep texture renderer can compute accurate gradients as the neighborhood vertices on the template 3D model are not strongly correlated. This local constraint is critical for fast and accurate pose refinement.

Furthermore, we perform additional experiments to verify the effect of feature warping. To this end, we warp the feature extracted by deep texture renderer based on the updated pose (Ours w/ FW). The result in Table 4 shows that Ours w/ FW achieves 9.2% absolute improvement from PVNet [27] on the LineMOD dataset [15]. However, Table 5 demonstrates the limited ability on the Occlusion LineMOD dataset [6]. From this result, we figure out that warping has an inferior influence on refinement of occluded objects. We conjecture that this difference comes from the fact that warping can not deal with large pose error because unlike our proposed RePOSE, feature warping can only consider the visible surface at the first step. Being different from the methods using feature warping, our iterative deep texture rendering method can generate a feature with a complete shape. We believe this characteristics of feature rendering leads to successful pose refinement.

Comparison with the latest refinement network on the LineMOD dataset. We compare our refinement network with the latest fully CNN-based refinement network proposed in the paper of DPOD [40]. In this experiment, we use the same initial pose estimator [27]. Since DPOD is fully CNN-based, we increased the amount of the dataset by twice. The refinement network of DPOD outputs a refined pose based on a cropped input RGB image and a synthetic rendering with an initial pose estimate. The experimental result in Tables 4 and 5 shows DPOD fails to refine pose well when trained with the small amount of the dataset. The refinement network of DPOD estimates a refined pose directly and do not consider projective geometry explicitly. This means their network needs to learn not only deep features but also mapping of the deep feature into an object's pose from training data. Several papers [7, 31, 8, 30] report that learning a less complex task can achieve better accuracy and generalization in a 6D camera localization task. Also, we assume the low ADD(-S) score on Occlusion LineMOD dataset implies its low generalization performance to occluded objects. Our network only trains deep features and a refined object's pose is acquired by solving minimization problem based on projective geometry. From this experimental result, we believe the same principle proposed in the field of 6D camera localization is still valid in 6D object pose estimation.

Table 6: Comparison of number of iterations and refinement runtime. ADD(-S) on the Occlusion LineMOD dataset is reported in this table. Our proposed network is trained by using a pose loss for 5 iterations.

Method	Iteration	ADD(-S) Score	Runtime
AAE [40]	-	-	200 ms
SSD6D [40]	-	-	24 ms
DPOD [40]	-	47.3	5 ms
Ours	0	40.8	0 ms
	1	45.7	4.1 ms
	2	48.6	5.8 ms
	3	50.1	7.5 ms
	4	51.0	9.2 ms
	5	51.6	10.9 ms

Number of iteration and run time analysis. Our proposed refinement network, RePOSE can adjust the trade-off between the accuracy and run time by changing the number of iterations. We show the ADD(-S) score and the run time on the Occlusion LineMOD dataset with each iteration count in Table 6. On a machine equipped with Nvidia RTX2080 Super GPU and Ryzen 7 3700X CPU, our method takes 1.7 ms per iteration (deep texture rendering + pose update through LM optimization [24]). This result shows our method achieves higher performance with the faster or comparable runtime than prior art.

5. Conclusion

Real-time pose estimation needs accurate and fast pose refinement. Our proposed method, RePOSE uses efficient deep texture renderer to perform pose refinement at 92 FPS and has practical applications as a real-time 6D object tracker. Our experiments show that learnable deep textures coupled with the efficient non-linear optimization results in accurate 6D object poses. Further, our ablations highlight the fundamental limitations of a convolutional neural network to extract critical information useful for pose refinement. We believe that the concept of using efficient renderers with learnable deep textures instead of a CNN for pose refinement is an important conceptual change and will inspire a new research direction for real-time 6D object pose estimation.

6. Acknowledgment

This work is funded in part by the Department of Homeland Security award 2017-DN-077-ER0001, JST AIP Acceleration, Grant Number JPMJCR20U1, and JST CREST, Grant Number JPMJCR19F5, Japan.

References

[1] Einscan pro 2x: https://www.einscan.com/handheld-3d-scanner/einscan-pro-2x/. 2

[2] Bundle adjustment - a modern synthesis. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, 2000. 7

[3] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Bundle adjustment in the large. In *ECCV*, 2010. 7

[4] Hatem Said Alismail, Brett Browning, and Simon Lucey. Photometric bundle adjustment for vision-based slam. In *ACCV*, 2016. 2

[5] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. In *IJCV*, 2004. 1

[6] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *ECCV*, 2014. 2, 5, 8

[7] Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. DSAC - Differentiable RANSAC for Camera Localization. In *CVPR*, 2016. 8

[8] Eric Brachmann and Carsten Rother. Learning less is more - 6d camera localization via 3d surface regression. In *CVPR*, 2018. 8

[9] Yannick Bukschat and Marcus Vetter. Efficientpose: An efficient, accurate and scalable end-to-end 6d multi object pose estimation approach. In *CoRR*, 2020. 6

[10] Berk Calli, Arjun Singh, Aaron Walsman, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. The ycb object and model set: Towards common benchmarks for manipulation research. In *ICAR*, 2015. 5

[11] Bo Chen, Alvaro Parra, Nan Cao, and Tat-Jun Chin. End-to-end learnable geometric vision by backpropagating pnp optimization. In *CVPR*, 2020. 2

[12] Ronald Clark, Michael Bloesch, Jan Czarnowski, Stefan Leutenegger, and Andrew J. Davison. Learning to solve nonlinear least squares for monocular stereo. In *ECCV*, 2018. 2

[13] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *ECCV*, 1998. 1

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 5

[15] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *ACCV*, 2012. 2, 5, 8

[16] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *CVPR*, 2018. 4

[17] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *ICCV*, 2017. 2

[18] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015. 4

[19] Y. Labbe, J. Carpentier, M. Aubry, and J. Sivic. Cosypose: Consistent multi-view multi-object 6d pose estimation. In *ECCV*, 2020. 1, 2, 6, 7

[20] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *IJCV*, 2009. 2

[21] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. In *ECCV*, 2018. 1, 2, 6

[22] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *IJCAI*, 1981. 1

[23] Iain Matthews and Simon Baker. Active appearance models revisited. Number CMU-RI-TR-03-02, Pittsburgh, PA, 2003. 1

[24] Jorge J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In *Numerical Analysis*, 1978. 1, 2, 3, 8

[25] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. In *T-RO*, 2015. 2

[26] Markus Oberweger, Mahdi Rad, and Vincent Lepetit. Making deep heatmaps robust to partial occlusions for 3d object pose estimation. In *ECCV*, 2018. 2

[27] Sida Peng, Xiaowei Liu, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *CVPR*, 2019. 2, 3, 5, 6, 7, 8

[28] Mahdi Rad and Vincent Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *ICCV*, 2017. 1, 2

[29] O. Ronneberger, P.Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, 2015. 3, 5

[30] Torsten Sattler, B. Leibe, and L. Kobbelt. Efficient & effective prioritized matching for large-scale image-based localization. *PAMI*, 2017. 8

[31] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixe. Understanding the limitations of cnn-based absolute camera pose regression. In *CVPR*, 2019. 8

[32] J. L. Schönberger and J. Frahm. Structure-from-motion revisited. In *CVPR*, 2016. 2

[33] Jiaru Song and Qixing Huang. Hybridpose: 6d object pose estimation under hybrid representations. In *CVPR*, 2020. 2, 6

[34] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3d orientation learning for 6d object detection from rgb images. In *ECCV*, 2018. 2

[35] Chengzhou Tang and Ping Tan. BA-Net: Dense Bundle Adjustment Network. *ICLR*, 2019. 2, 3

[36] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. In *CoRL*, 2018. 2

[37] L. von Stumberg, P. Wenzel, Q. Khan, and D. Cremers. GN-Net: The Gauss-Newton Loss for Multi-Weather Relocalization. In *ICRA*, 2020. 2

[38] Angtian Wang, Adam Kortylewski, and Alan Yuille. Nemo: Neural mesh models of contrastive features for robust 3d pose estimation. In *ICLR*, 2021. 2

[39] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. In *RSS*, 2018. 2, 3, 5, 6

[40] Sergey Zakharov, Ivan Shugurov, and Slobodan Ilic. Dpod: 6d pose object detector and refiner. In *ICCV*, 2019. 1, 2, 6, 8

[41] Zhengyou Zhang. Iterative Closest Point (ICP). In *Computer Vision: A Reference Guide*, 2014. 2