Supplementary Material for EPro-PnP: Generalized End-to-End Probabilistic Perspective-n-Points for Monocular Object Pose Estimation

Hansheng Chen,[1,2,*] Pichao Wang,[2,†] Fan Wang,[2] Wei Tian,[1,†] Lu Xiong,[1] Hao Li[2]
[1]School of Automotive Studies, Tongji University [2]Alibaba Group
hanshengchen97@gmail.com {tian_wei, xiong_lu}@tongji.edu.cn
{pichao.wang, fan.w, lihao.lh}@alibaba-inc.com

A. Levenberg-Marquardt PnP Solver

For scalability, we have implemented a PyTorch-based batch Levenberg-Marquardt (LM) PnP solver. The implementation generally follows the Ceres solver [1]. Here, we discuss some important details that are related to the proposed Monte Carlo pose sampling and derivative regularization.

A.1. Adaptive Huber Kernel

To robustify the weighted reprojection errors of various scales, we adopt an adaptive Huber kernel with a dynamic threshold δ for each object, defined as a function of the weights w_i^{2D} and 2D coordinates x_i^{2D}:

$$\delta = \delta_{\mathrm{rel}} \frac{\left\|\bar{w}^{\mathrm{2D}}\right\|_1}{2} \left(\frac{1}{N-1} \sum_{i=1}^{N} \left\| x_i^{\mathrm{2D}} - \bar{x}^{\mathrm{2D}} \right\|^2 \right)^{\frac{1}{2}}, \tag{13}$$

with the relative threshold δ_{rel} as hyperparameter, and the mean vectors $\bar{w}^{\mathrm{2D}} = \frac{1}{N}\sum_{i=1}^{N} w_i^{\mathrm{2D}}$, $\bar{x}^{\mathrm{2D}} = \frac{1}{N}\sum_{i=1}^{N} x_i^{\mathrm{2D}}$.

A.2. LM Step with Huber Kernel

Adding the Huber kernel influences every related element from the likelihood function to the LM iteration step and derivative regularization loss. Thanks to PyTorch's automatic differentiation, the robustified Monte Carlo KL divergence loss does not require much special handling. For the LM solver, however, the residual $F(y)$ (concatenated weighted reprojection errors) and the Jacobian matrix J have to be rescaled before computing the robustified LM step [14].

The rescaled residual block $\tilde{f}_i(y)$ and Jacobian block $\tilde{J}_i(y)$ of the i-th point pair are defined as:

$$\tilde{f}_i(y) = \sqrt{\rho'_i} f_i(y), \tag{14}$$

$$\tilde{J}_i(y) = \sqrt{\rho'_i} J_i(y), \tag{15}$$

where

$$\rho'_i = \begin{cases} 1, & \|f_i(y)\| \leq \delta, \\ \dfrac{\delta}{\|f_i(y)\|}, & \|f_i(y)\| > \delta, \end{cases} \tag{16}$$

$$J_i(y) = \frac{\partial f_i(y)}{\partial y^{\mathrm{T}}}. \tag{17}$$

Following the implementation of Ceres solver [1], the robustified LM iteration step is:

$$\Delta y = -\left(\tilde{J}^{\mathrm{T}}\tilde{J} + \lambda D^2\right)^{-1} \tilde{J}^{\mathrm{T}}\tilde{F}, \tag{18}$$

where

$$\tilde{J} = \begin{bmatrix} \tilde{J}_1(y) \\ \vdots \\ \tilde{J}_N(y) \end{bmatrix}, \tilde{F} = \begin{bmatrix} \tilde{f}_1(y) \\ \vdots \\ \tilde{f}_N(y) \end{bmatrix}, \tag{19}$$

D is the square root of the diagonal of the matrix $\tilde{J}^{\mathrm{T}}\tilde{J}$, and λ is the reciprocal of the LM trust region radius [1].

Note that the rescaled residual and Jacobian affects the derivative regularization (Eq. (10)), as well as the covariance estimation in the next subsection.

Fast Inference Mode We empirically found that in a well-trained model, the LM trust region radius can be initialized with a very large value, effectively rendering the LM algorithm redundant. We therefore use the simple Gauss-Newton implementation for fast inference:

$$\Delta y = -\left(\tilde{J}^{\mathrm{T}}\tilde{J} + \varepsilon I\right)^{-1} \tilde{J}^{\mathrm{T}}\tilde{F}, \tag{20}$$

where ε is a small value for numerical stability.

A.3. Covariance Estimation

During training, the concentration of the AMIS proposal is determined by the local estimation of pose covariance matrix Σ_{y^*}, defined as:

$$\Sigma_{y^*} = \left(\tilde{J}^{\mathrm{T}}\tilde{J} + \varepsilon I\right)^{-1}\Big|_{y=y^*}, \tag{21}$$

*Part of work done during an internship at Alibaba Group.
†Corresponding authors: Pichao Wang, Wei Tian.

where y^* is the LM solution that determines the location of the proposal distribution.

A.4. Initialization

Since the LM solver only finds a local solution, initialization plays a determinant role in dealing with ambiguity. Standard EPnP [11] initialization can handle the dense correspondence network trained on the LineMOD [9] dataset, where ambiguity is not noticeable. For the deformable correspondence network trained on the nuScenes [4] dataset and more general cases, we implement a random sampling algorithm analogous to RANSAC, to search for the global optimum efficiently.

Given the N-point correspondence set $X = \left\{x_i^{\mathrm{3D}}, x_i^{\mathrm{2D}}, w_i^{\mathrm{2D}} \,\middle|\, i = 1 \cdots N\right\}$, we generate M subsets consisting of n corresponding points each ($3 \leq n < N$), by repeatedly sub-sampling n indices without replacement from a multinomial distribution, whose probability mass function $p(i)$ is defined by the corresponding weights:

$$p(i) = \frac{\left\|w_i^{\mathrm{2D}}\right\|_1}{\sum_{i=1}^{N} \left\|w_i^{\mathrm{2D}}\right\|_1}. \tag{22}$$

From each subset, a pose hypothesis can be solved via the LM algorithm with very few iterations (we use 3 iterations). This is implemented as a batch operation on GPU, and is rather efficient for small subsets. We take the hypothesis of maximum log-likelihood $\log p(X|y)$ as the initial point, starting from which subsequent LM iterations are computed on the full set X.

Training Mode During training, the LM PnP solver is utilized for estimating the location and concentration of the initial proposal distribution in the AMIS algorithm. The location is very important to the stability of Monte Carlo training. If the LM solver fails to find the global optimum and the location of the local optimum is far from the true pose y_{gt}, the balance between the two opposite signed terms in Eq. (5) may be broken, leading to exploding gradient in the worst case scenario. To avoid such problem, we adopt a simple initialization trick: we compare the log-likelihood $\log p(X|y)$ of the ground truth y_{gt} and the selected hypothesis, and then keep the one with higher likelihood as the initial state of the LM solver.

B. Details on Monte Carlo Pose Sampling

B.1. Proposal Distribution for Position

For the proposal distribution of the translation vector $t \in \mathbb{R}^3$, we adopt the multivariate t-distribution, with the following probability density function (PDF):

$$q_{\mathrm{T}}(t) = \frac{\Gamma\left(\frac{\nu+3}{2}\right)}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\nu^3\pi^3|\Sigma|}} \left(1 + \frac{1}{\nu}\|t-\mu\|_\Sigma^2\right)^{-\frac{\nu+3}{2}}, \tag{23}$$

where $\|t-\mu\|_\Sigma^2 = (t-\mu)^{\mathrm{T}}\Sigma^{-1}(t-\mu)$, with the location μ, the 3×3 positive definite scale matrix Σ, and the degrees of freedom ν. Following [6], we set ν to 3. Compared to the multivariate normal distribution, the t-distribution has a heavier tail, which is ideal for robust sampling.

The multivariate t-distribution has been implemented in the Pyro [2] package.

Initial Parameters The initial location and scale is determined by the PnP solution and covariance matrix, *i.e.*, $\mu \leftarrow t^*, \Sigma \leftarrow \Sigma_{t^*}$, where Σ_{t^*} is the 3×3 submatrix of the full pose covariance Σ_{p^*}. Note that the actual covariance of the t-distribution is thus $\frac{\nu}{\nu-1}\Sigma_{t^*}$, which is intentionally scaled up for robust sampling in a wider range.

Parameter Estimation from Weighted Samples To update the proposal, we let the location μ and scale Σ be the first and second moment of the weighted samples (*i.e.*, weighted mean and covariance), respectively.

B.2. Proposal Distribution for 1D Orientation

For the proposal distribution of the 1D yaw-only orientation θ, we adopt a mixture of von Mises and uniform distribution. The von Mises is also known as the circular normal distribution, and its PDF is given by:

$$q_{\mathrm{VM}}(\theta) = \frac{\exp\left(\kappa\cos\left(\theta-\mu\right)\right)}{2\pi I_0(\kappa)}, \tag{24}$$

where μ is the location parameter, κ is the concentration parameter, and $I_0(\cdot)$ is the modified Bessel function with order zero. The mixture PDF is thus:

$$q_{\mathrm{mix}}(\theta) = (1-\alpha)q_{\mathrm{VM}}(\theta) + \alpha q_{\mathrm{uniform}}(\theta), \tag{25}$$

with the uniform mixture weight α. The uniform component is added in order to capture other potential modes under orientation ambiguity. We set α to a fixed value of $1/4$.

PyTorch has already implemented the von Mises distribution, but its random sample generation is rather slow. As an alternative we use the NumPy implementation for random sampling.

Initial Parameters With the yaw angle θ^* and its variance $\sigma_{\theta^*}^2$ from the PnP solver, the parameters of the von Mises proposal is initialized by $\mu \leftarrow \theta^*, \kappa \leftarrow \frac{1}{3\sigma_{\theta^*}^2}$.

Parameter Estimation from Weighted Samples For the location μ, we simply adopt its maximum likelihood estimation, *i.e.*, the circular mean of the weighted samples. For the concentration κ, we first compute an approximated estimation [7] by:

$$\hat{\kappa} = \frac{\bar{r}(2-\bar{r}^2)}{1-\bar{r}^2}, \tag{26}$$

where $\bar{r} = \left\|\sum_j v_j[\sin\theta_j, \cos\theta_j]^{\mathrm{T}} / \sum_j v_j\right\|$ is the norm of the mean orientation vector, with the importance weight v_j for the j-th sample θ_j. Finally, the concentration is scaled down for robust sampling, such that $\kappa \leftarrow \hat{\kappa}/3$.

B.3. Proposal Distribution for 3D Orientation

Regarding the quaternion based parameterization of 3D orientation, which can be represented by a unit 4D vector l, we adopt the angular central Gaussian (ACG) distribution as the proposal. The support of the 4-dimensional ACG distribution is the unit hypersphere, and the PDF is given by:

$$q_{\mathrm{ACG}}(l) = \frac{(l^{\mathrm{T}}\Lambda^{-1}l)^{-2}}{S_4|\Lambda|^{\frac{1}{2}}}, \tag{27}$$

where $S_4 = 2\pi^2$ is the 3D surface area of the 4D sphere, and Λ is a 4×4 positive definite matrix.

The ACG density can be derived by integrating the zero-mean multivariate normal distribution $\mathcal{N}(0, \Lambda)$ along the radial direction from 0 to $\inf$. Therefore, drawing samples from the ACG distribution is equivalent to sampling from $\mathcal{N}(0, \Lambda)$ and then normalizing the samples to unit radius.

Initial Parameters Consider l^* to be the PnP solution and $\Sigma_{l^*}^{-1}$ to be the estimated 4×4 inverse covariance matrix. Note that $\Sigma_{l^*}^{-1}$ is only valid in the local tangent space with rank 3, satisfying $l^{*\mathrm{T}}\Sigma_{l^*}^{-1}l^* = 0$. The initial parameters are determined by:

$$\Lambda \leftarrow \hat{\Lambda} + \alpha|\hat{\Lambda}|^{\frac{1}{4}}I, \tag{28}$$

where $\hat{\Lambda} = \left(\Sigma_{l^*}^{-1} + I\right)^{-1}$, and α is a hyperparameter that controls the dispersion of the proposal for robust sampling. We set α to 0.001 in the experiments.

Parameter Estimation from Weighted Samples Based on the samples l_j and weights v_j, the maximum likelihood estimation $\hat{\Lambda}$ is the solution to the following equation:

$$\hat{\Lambda} = \frac{4}{\sum_j v_j} \sum_j \frac{v_j l_j l_j^{\mathrm{T}}}{l_j^{\mathrm{T}}\hat{\Lambda}^{-1}l_j}. \tag{29}$$

The solution to Eq. (29) can be computed by fixed-point iteration [15]. The final parameters of the updated proposal is determined the same way as in Eq. (28).

C. Details on Derivative Regularization Loss

As stated in the main paper, the derivative regularization loss L_{reg} consists of the position loss L_{pos} and the orientation loss L_{orient}.

For L_{pos}, we adopt the smooth L1 loss based on the Euclidean distance $d_t = \|t^* + \Delta t - t_{\mathrm{gt}}\|$, given by:

$$L_{\mathrm{pos}} = \begin{cases} \frac{d_t^2}{2\beta}, & d_t \le \beta, \\ d_t - 0.5\beta, & d_t > \beta, \end{cases} \tag{30}$$

with the hyperparameter β.

For L_{orient}, we adopt the cosine similarity loss based on the angular distance d_θ. For 1D orientation parameterized by the angle θ, $d_\theta = \theta^* + \Delta\theta - \theta_{\mathrm{gt}}$. For 3D orientation parameterized by the quaternion vector l, $d_\theta = 2\arccos{(l^* + \Delta l)^{\mathrm{T}} l_{\mathrm{gt}}}$. The loss function is therefore defined as:

$$L_{\mathrm{orient}} = 1 - \cos d_\theta. \tag{31}$$

For 3D orientation, after the substitution, the loss function can be simplified to:

$$L_{\mathrm{orient}} = 2 - 2\left((l^* + \Delta l)^{\mathrm{T}} l_{\mathrm{gt}}\right)^2. \tag{32}$$

For the specific settings of the hyperparameter β and loss weights, please refer to the experiment configuration code.

D. Details on the Deformable Correspondence Network

D.1. Network Architecture

The detailed network architecture of the deformable correspondence network is shown in Figure 8. Following deformable DETR [17], this paper adopts the multi-head deformable sampling. Let n_{head} be the number of heads and n_{hpts} be the number of points per head, a total number of $N = n_{\mathrm{head}}n_{\mathrm{hpts}}$ points are sampled for each object. The sampling locations relative to the reference point are generated from the object embedding by a single layer of linear transformation. We set n_{head} to 8, which yields $256/n_{\mathrm{head}} = 32$ channels for the point features.

The point-level branch on the left side of Figure 8 is responsible for predicting the 3D points x_i^{3D} and corresponding weights w_i^{2D}. The sampled point features are first enhanced by the object-level context, by adding the reshaped head-wise object embedding to the point features. Then, the features of the N points are processed by the self attention layer, for which the 2D points are transformed into the Q-K vectors of positional information. The attention layer is followed by standard layers of normalization, skip connection, and feedforward network (FFN).

Regarding the object-level branch on the right side of Figure 8, a multi-head attention layer is employed to aggregate the sampled point features. Unlike the original deformable attention layer [17] that predicts the attention weights by linear projection of the object embedding, we adopt the full Q-K dot-product attention with positional encoding. After being processed by the subsequent layers, the object-level features are finally transformed into to the object-level predictions, consisting of the 3D localization score, weight scale, 3D bounding box size, and other optional properties (velocity and attribute). Note that the attention layer is actually not a necessary component for object-level predictions, but rather a byproduct of the deformable point samples whose features can be leveraged with little computation overhead.

D.2. Loss Functions for Object-Level Predictions

As in FCOS3D [16], we adopt smooth L1 regression loss for 3D box size and velocity, and cross-entropy classification loss for attribute. Additionally, a binary cross-entropy loss is imposed upon the 3D localization score, with the target c_{tgt} defined as a function of the position error:

$$\begin{aligned} c_{\text{tgt}} &= \mathit{Score}(\|t^*_{\text{XZ}} - t_{\text{XZgt}}\|) \\ &= \max(0, \min(1, -a \log \|t^*_{\text{XZ}} - t_{\text{XZgt}}\| + b)), \end{aligned} \tag{33}$$

where t^*_{XZ} is the XZ components of the PnP solution, t_{XZgt} is the XZ components of the true pose, and a, b are the linear coefficients. The predicted 3D localization score c_{pred} shall reflect the positional uncertainty of an object, as a faster alternative to evaluating the uncertainty via the Monte Carlo method during inference (Section F.2). The final detection score is defined as the product of the predicted 3D score and the classification score from the base detector.

D.3. Auxiliary Loss Functions

To regularize the dense features, we append an auxiliary branch that predicts the multi-head dense 3D coordinates and corresponding weights, as shown in Figure 9. Leveraging the ground truth of object 2D boxes, the features within the box regions are densely sampled via RoI Align [8], and transformed into the 3D coordinates $x^{3\text{D}}$ and weights $w^{2\text{D}}$ via an independent linear layer. Besides, the attention weights ϕ are obtained via Q-K dot-product and normalized along the n_{head} dimension and across the overlapping region of multiple RoIs via Softmax.

During training, we impose the reprojection-based auxiliary loss for the multi-head dense predictions, formulated as the negative log-likelihood (NLL) of the Gaussian mixture model [3]. The loss function for each sampled point is defined as:

$$L_{\text{proj}} = -\log \sum_{\text{RoI}} \sum_{k=1}^{n_{\text{head}}} \phi_k |\operatorname{diag} w_k^{2\text{D}}| \exp -\frac{1}{2} \|f_k(y_{\text{gt}})\|^2, \tag{34}$$

where k is the head index, $f_k(y_{\text{gt}})$ is the weighted reprojection error of the k-th head at the truth pose y_{gt}. In the above equation, the diagonal matrix $\operatorname{diag} w_k^{2\text{D}}$ is interpreted as the inverse square root of the covariance matrix of the normal distribution, *i.e.*, $\operatorname{diag} w_k^{2\text{D}} = \Sigma^{-\frac{1}{2}}$, and the head attention weight ϕ_k is interpreted as the mixture component weight. $\sum_{\text{RoI}}$ is a special operation that takes the overlapping region of multiple RoIs into account, formulating a mixture of multiple heads and multiple RoIs (see code for details).

Another auxiliary loss is the coordinate regression loss that introduces the geometric knowledge. Following MonoRUn [5], we extract the sparse ground truth of 3D coordinates $x^{3\text{D}}_{\text{gt}}$ from the 3D LiDAR point cloud. The multi-head coordinate regression loss for each sampled point with available ground truth is defined as:

$$L_{\text{regr}} = \sum_{k=1}^{n_{\text{head}}} \phi_k \rho\Big(\big\|x_k^{3\text{D}} - x_{\text{gt}}^{3\text{D}}\big\|^2\Big), \tag{35}$$

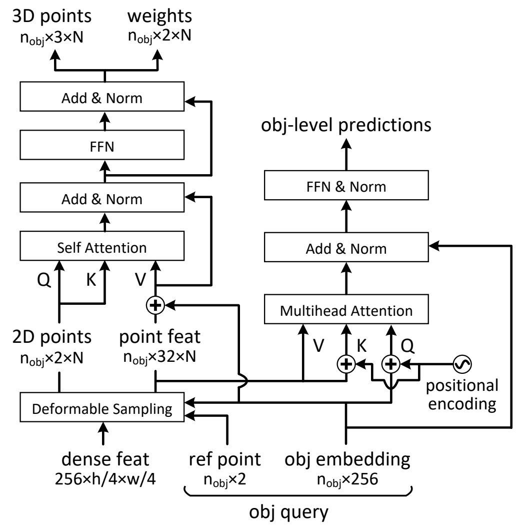

Figure 8. Detailed architecture of the deformable correspondence network.

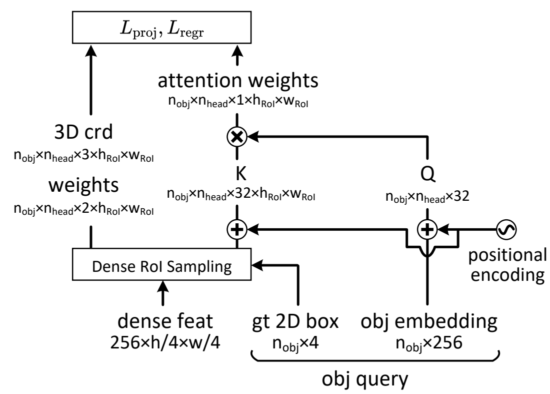

Figure 9. Architecture of the auxiliary branch. This branch shares the same weights of Q, K projection with the deformable attention layer in the lower right of Figure 8.

where $\rho(\cdot)$ is the Huber kernel. L_{regr} is essentially a weighted smooth L1 loss (although we write the Huber kernel for convenience in notation).

D.4. Training Strategy

During training, we randomly sample 48 positive object queries from the FCOS3D [16] detector for each image, which limits the batch size of the deformable correspondence network to control the computation overhead of the Monte Carlo pose loss.

E. Additional Results of the Dense Correspondence Network

E.1. Convergence Behavior

The convergence behaviors of EPro-PnP and CDPN [12] are compared in Figure 10. The original CDPN-Full is

ID	Method	Data	NDS	mAP	True positive metrics (lower is better)				
					mATE	mASE	mAOE	mAVE	mAAE
A0	Basic EPro-PnP	Val	0.425	0.349	0.676	0.263	0.363	1.035	0.196
A1	A0 + coord. regr.	Val	0.430	0.352	0.667	0.258	0.337	1.031	0.193
B0	A0 $\rightarrow$ No reprojection L_{proj}	Val	0.408	0.337	0.721	0.267	0.452	1.113	0.166
C0	A0 $\rightarrow$ 50% Monte Carlo score	Val	0.424	0.350	0.673	0.264	0.373	1.042	0.198
C1	A0 $\rightarrow$ 100% Monte Carlo score	Val	0.424	0.350	0.675	0.264	0.367	1.048	0.199
D0	A1 $\rightarrow$ Compact network	Val	0.434	0.358	0.672	0.264	0.351	0.983	0.181
D1	D0 + TTA	Val	0.446	0.367	0.664	0.260	0.320	0.951	0.179

Table 5. Additional results of the deformable correspondence network tested on the nuScenes [4] benchmark.

trained in 3 stages (rotation head – translation head – both together) with a total of 480 epochs. In contrast, EPro-PnP with derivative regularization clearly outperforms CDPN-Full within one stage, and goes further when initialized from the pretrained first-stage CDPN.

E.2. Inference Time

Compared to the inference pipeline of CDPN-Full [12], EPro-PnP does not use the RANSAC algorithm or extra translation head, so the overall inference speed is more than twice as fast as CDPN-Full (at a batch size of 32), even though we introduces the iterative LM solver.

Regarding the LM solver itself, inference takes 7.3 ms for a batch of one object, measured on RTX 2080 Ti GPU, excluding EPnP [11] initialization. As a reference, the state-of-the-art pose refiner RePOSE [10] (also based on the LM algorithm) adds 10.9 ms overhead to the base pose estimator PVNet [13] at the same batch size, measured on RTX 2080 Super GPU, which is slower than ours. Nevertheless, faster inference is possible if the number of points $N = 64 \times 64$ is reduced to an optimal level.

F. Additional Experiments on the Deformable Correspondence Network

F.1. On the Auxiliary Reprojection Loss

As shown in Table 5, removing the auxiliary reprojection loss in Eq. 34 lowers the 3D object detection accuracy (NDS 0.408 *vs.* 0.425). Among the true positive metrics, the orientation metric mAOE is the most affected. The results indicate that, although the deformable correspondences can be learned solely with the end-to-end loss, it is still beneficial to add auxiliary task for further regularization, even if the task itself does not involve extra annotation.

F.2. On the Uncertainty of Object Pose

The dispersion of the inferred pose distribution reflects the aleatoric uncertainty of the predicted pose. Previous work [5] reasons the pose uncertainty by propagating the reprojection uncertainty learned from a surrogate loss through the PnP operation, but that uncertainty requires calibration

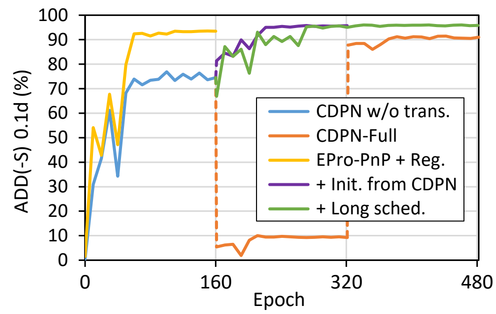

Figure 10. Testing accuracy *vs.* training progress on LineMOD.

and is not reliable enough. In our work, the pose uncertainty is learned with the KL-divergence-based pose loss in an end-to-end manner, which is much more reliable in theory.

To quantitatively evaluate the reliability of the pose uncertainty in terms of measuring the localization confidence, a straightforward approach is to compute the 3D localization score c_{MC} via Monte Carlo pose sampling, and compare the resulting mAP against the standard implementation with 3D score c_{pred} predicted from the object-level branch. With the PnP solution t^*, the sampled translation vector t_j, and its importance weight v_j, the Monte Carlo score is computed by:

$$c_{\text{MC}} = \frac{1}{\sum_j v_j} \sum_j v_j \mathit{Score}\left(\|t^*_{\text{XZ}} - t_{\text{XZ}j}\|\right), \qquad (36)$$

where the subscript $(\cdot)_{\text{XZ}}$ denotes taking the XZ components, and the function $\mathit{Score}(\cdot)$ is the same as in Eq. 33. Furthermore, the final score can also be a mixture of the two sources, defined as:

$$c_{\text{mix}} = c_{\text{MC}}^{\alpha} c_{\text{pred}}^{1-\alpha}, \qquad (37)$$

where α is the mixture weight.

The evaluation results under different mixture weights are presented in Table 5. Regarding the mAP metric, the Monte Carlo score is on par with the standard implementation (0.350 *vs.* 0.350 *vs.* 0.349), indicating that the pose uncertainty is a reliable measure of the detection confidence.

Nevertheless, due to the much longer runtime of inferring with Monte Carlo pose sampling, training a standard score branch is still a more practical choice.

F.3. On the Network Redundancy and Potential for Future Improvement

Since the main concern of this paper is to propose a novel differentiable PnP layer, we did not have enough time and resources to fine-tune the architecture and parameters of the deformable correspondence network at the time of submitting the manuscript. Therefore, the network described in Sections 4.2 and D.1 was crafted with some redundancy in mind, being not very efficient in terms of FLOP count, memory footprint and inference time, leaving large potential for improvement.

To demonstrate the potential for improvement, we train a more compact network with lower resolution (`stride=8`) for the dense feature map, and the number of points per head n_{hpts} reduced from 32 to 16, and squeeze the batch of 12 images into 2 RTX 3090 GPUs. As shown in Table 5, the overall performance is actually slightly better than the original version (NDS 0.434 *vs*. 0.430). Still, a more efficient architecture is yet to be determined in future work.

Inference Time Regarding the compact network, the average inference time per frame (comprising a batch of 6 surrounding 1600×672[5] images, without TTA) is shown in Table 6, measured on RTX 3090 GPU and Core i9-10920X CPU. On average, the batch PnP solver processes 625.97 objects per frame before non-maximum suppression (NMS).

PyTorch	Backbone & FPN	Heads		PnP	Total
		FCOS	Deform		
v1.8.1+cu111	0.195	0.074	0.028	**0.026**	0.327
v1.10.1+cu113	0.172	0.056	0.025	0.045	**0.301**

Table 6. Inference time (sec) of the deformable correspondence network on nuScenes object detection dataset [4]. The PnP solver (including the random sampling initialization in Section A.4) works faster (26 ms) with PyTorch v1.8.1, for which the code was originally developed, while the full model works faster (301 ms) with PyTorch v1.10.1.

G. Limitation

EPro-PnP is a versatile pose estimator for general problems, yet it has to be acknowledged that training the network with the Monte Carlo pose loss is inevitably *slower* than the baseline. At the batch size of 32, training the CDPN (without translation head) takes 143 seconds per epoch with the original coordinate regression loss, and 241 seconds per epoch with the Monte Carlo pose loss, which is about 70% longer time, as measured on GTX 1080 Ti GPU. However, the training time can be controlled by adjusting the number of Monte Carlo samples or the number of 2D-3D corresponding points. In this paper, the choice of these hyperparameters generally leans towards redundancy.

H. Additional Visualization

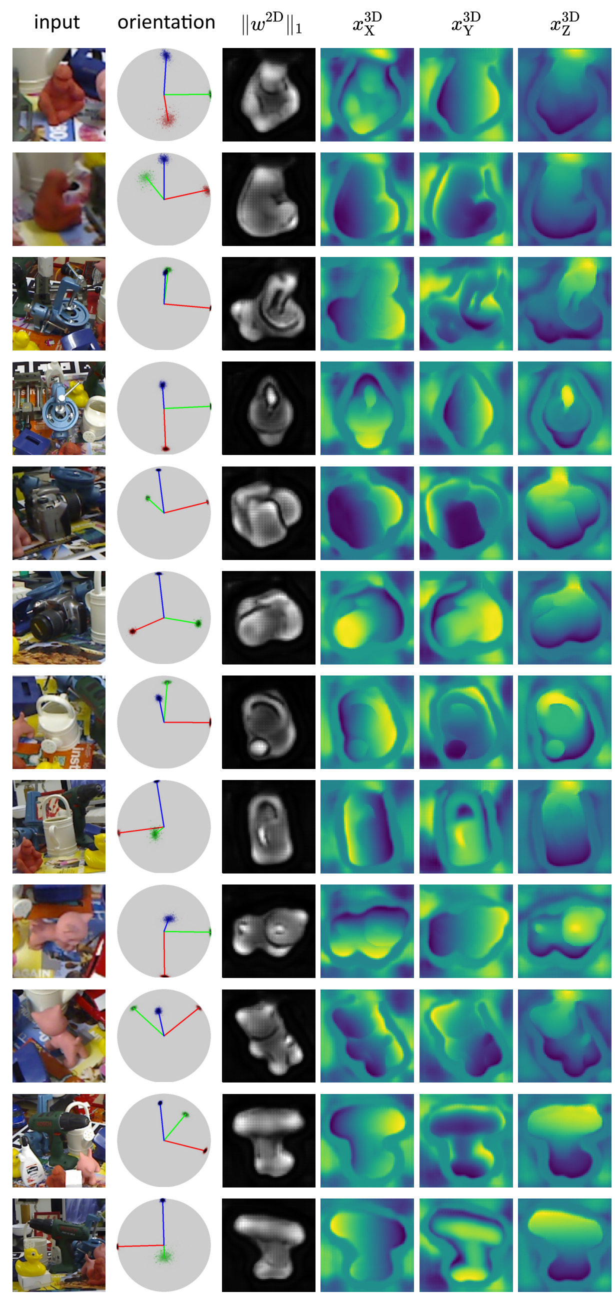

Figure 11. Inferred results on LineMOD test set by EPro-PnP with derivative regularization and pretrained CDPN weights, Part I.

[5] The original size is 1600×900. We crop the images for efficiency.

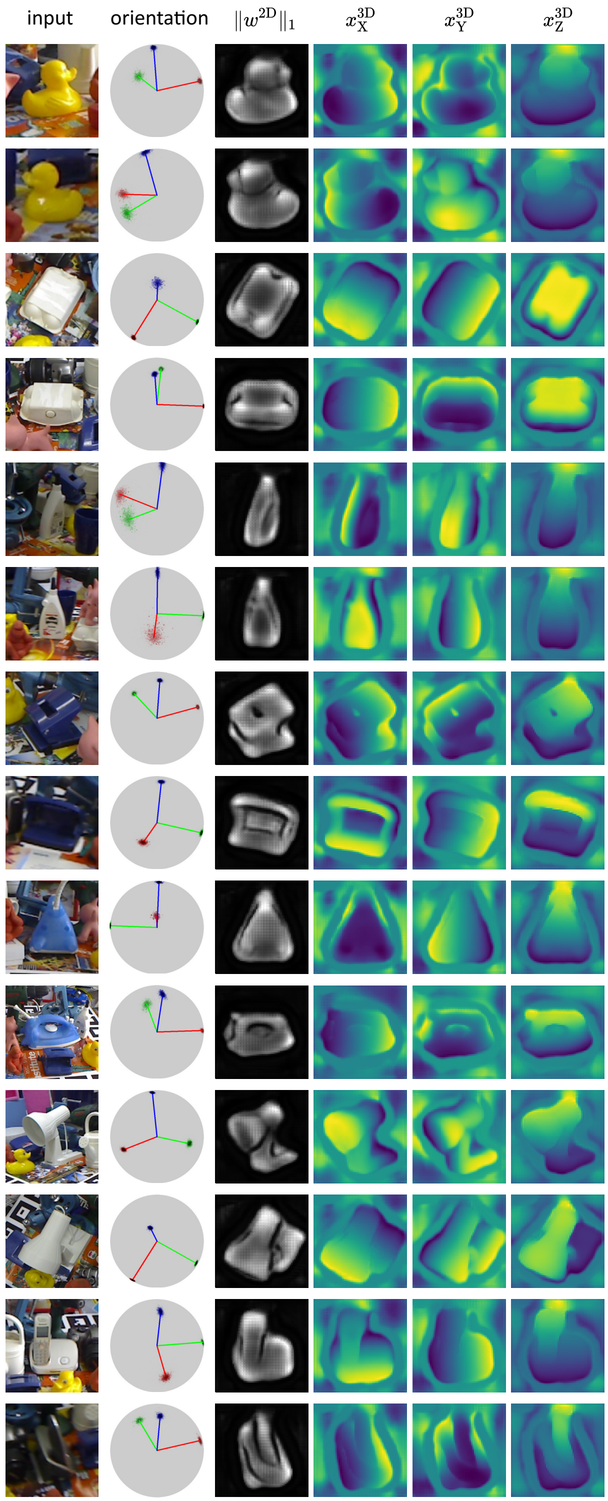

Figure 12. Inferred results on LineMOD test set by EPro-PnP with derivative regularization and pretrained CDPN weights, Part II.

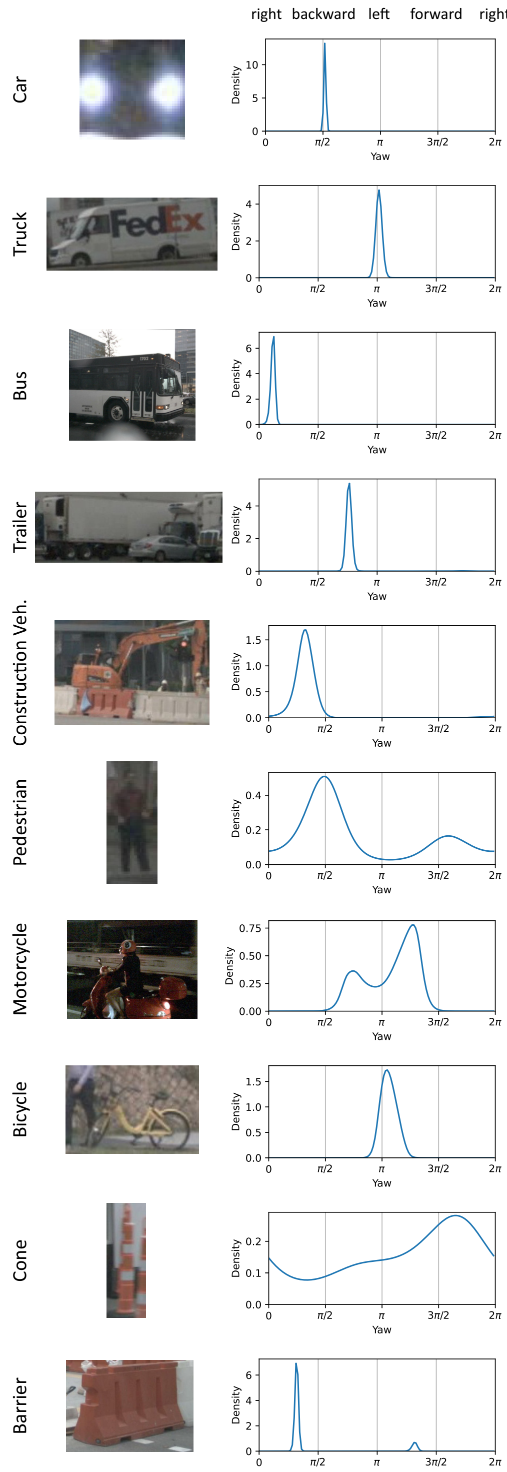

Figure 13. Inferred orientation on nuScenes validation set by the Basic EPro-PnP.

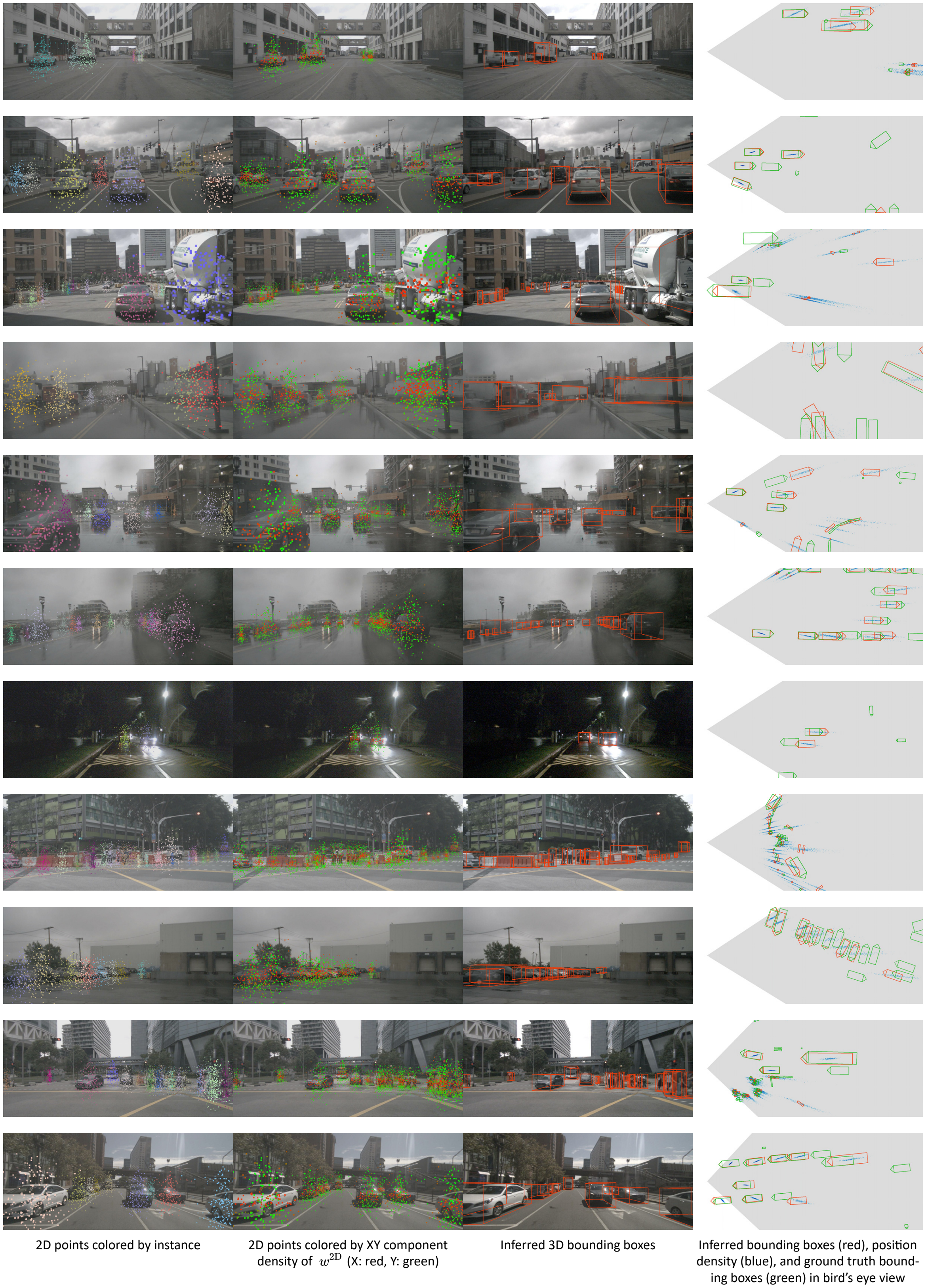

Figure 14. Inferred results on nuScenes validation set by the Basic EPro-PnP.

I. Notation

Notation		Description
x_i^{3D}	$\in \mathbb{R}^3$	Coordinate vector of the i-th 3D object point
x_i^{2D}	$\in \mathbb{R}^2$	Coordinate vector of the i-th 2D image point
w_i^{2D}	$\in \mathbb{R}_+^2$	Weight vector of the i-th 2D-3D point pair
X		The set of weighted 2D-3D correspondences
y		Object pose
y_{gt}		Ground truth of object pose
y^*		Object pose estimated by the PnP solver
R		3×3 rotation matrix representation of object orientation
θ		1D yaw angle representation of object orientation
l		Unit quaternion representation of object orientation
t	$\in \mathbb{R}^3$	Translation vector representation of object position
Σ_{y^*}		Pose covariance estimated by the PnP solver
J		Jacobian matrix
$\tilde{J}$		Rescaled Jacobian matrix
F		Concatenated vector of weighted reprojection errors of all points
$\tilde{F}$		Concatenated vector of rescaled weighted reprojection errors of all points
$\pi(\cdot)$	$: \mathbb{R}^3 \to \mathbb{R}^2$	Camera projection function
$f_i(y)$	$\in \mathbb{R}^2$	Weighted reprojection error of the i-th correspondence at pose y
$r_i(y)$	$\in \mathbb{R}^2$	Unweighted reprojection error of the i-th correspondence at pose y
$\rho(\cdot)$		Huber kernel function
ρ'_i		The derivative of the Huber kernel function of the i-th correspondence
δ		The Huber threshold
$p(X\|y)$		Likelihood function of object pose
$p(y)$		PDF of the prior pose distribution
$p(y\|X)$		PDF of the posterior pose distribution
$t(y)$		PDF of the target pose distribution
$q(y), q_t(y)$		PDF of the proposal pose distribution (of the t-th AMIS iteration)
y_j, y_j^t		The j-th random pose sample (of the t-th AMIS iteration)
v_j, v_j^t		Importance weight of the j-th pose sample (of the t-th AMIS iteration)
i		Index of 2D-3D point pair
j		Index of random pose sample
t		Index of AMIS iteration
N		Number of 2D-3D point pairs in total
K		Number of pose samples in total
T		Number of AMIS iterations
K'		Number of pose samples per AMIS iteration
n_{head}		Number of heads in the deformable correspondence network
n_{hpts}		Number of points per head in the deformable correspondence network
L_{KL}		KL divergence loss for object pose
L_{tgt}		The component of L_{KL} concerning the reprojection errors at target pose
L_{pred}		The component of L_{KL} concerning the reprojection errors over predicted pose
L_{reg}		Derivative regularization loss

Table 7. A summary of frequently used notations.

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org. 1

[2] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018. 2

[3] Christopher M. Bishop. Mixture density networks, 1994. 4

[4] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020. 2, 5, 6

[5] Hansheng Chen, Yuyao Huang, Wei Tian, Zhong Gao, and Lu Xiong. Monorun: Monocular 3d object detection by reconstruction and uncertainty propagation. In *CVPR*, 2021. 4, 5

[6] Jean-Marie Cornuet, Jean-Michel Marin, Antonietta Mira, and Christian P. Robert. Adaptive multiple importance sampling. *Scandinavian Journal of Statistics*, 39(4):798–812, 2012. 2

[7] Inderjit S. Dhillon and Suvrit Sra. Modeling data using directional distributions, 2003. 2

[8] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. 4

[9] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *ICCV*, 2011. 2

[10] Shun Iwase, Xingyu Liu, Rawal Khirodkar, Rio Yokota, and Kris M. Kitani. Repose: Fast 6d object pose refinement via deep texture rendering. In *ICCV*, 2021. 5

[11] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal Of Computer Vision*, 81:155–166, 2009. 2, 5

[12] Zhigang Li, Gu Wang, and Xiangyang Ji. Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation. In *ICCV*, 2019. 4, 5

[13] Sida Peng, Yuan Liu, Qixing Huang, Xiaowei Zhou, and Hujun Bao. Pvnet: Pixel-wise voting network for 6dof pose estimation. In *CVPR*, 2019. 5

[14] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment: A modern synthesis. In *International Workshop on Vision Algorithms: Theory and Practice*, 2000. 1

[15] David E. Tyler. Statistical analysis for the angular central gaussian distribution on the sphere. *Biometrika*, 74(3):579–589, 1987. 3

[16] Tai Wang, Xinge Zhu, Jiangmiao Pang, and Dahua Lin. FCOS3D: Fully convolutional one-stage monocular 3d object detection. In *ICCV Workshops*, 2021. 3, 4

[17] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. In *ICLR*, 2021. 3