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 $\delta$ for each object, defined as a function of the weights $w_i^{2D}$ and 2D coordinates $x_i^{2D}$:

$$
\delta = \delta_{rel} \left( \frac{1}{N} \sum_{i=1}^{N} \left\| w_i^{2D} \right\| \right) \left( \frac{1}{N} \sum_{i=1}^{N} \left\| x_i^{2D} \right\|^2 \right)^{\frac{1}{2}},
$$

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

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

$$
\tilde{r}_i(y) = \sqrt{\rho_i} r_i(y),
$$

$$
\tilde{J}_i(y) = \sqrt{\rho_i} J_i(y),
$$

where

$$
\rho_i = \begin{cases} 
1, & \|f_i(y)\| \leq \delta, \\
\frac{\delta}{\|f_i(y)\|}, & \|f_i(y)\| > \delta,
\end{cases}
$$

$$
J_i(y) = \frac{\partial f_i(y)}{\partial y}. 
$$

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

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

where

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

$D$ is the square root of the diagonal of the matrix $J^T J$, and $\lambda$ 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:

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

where $\varepsilon$ 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 $\Sigma_{y^*}$, defined as:

$$
\Sigma_{y^*} = \left( J^T J + \varepsilon I \right)^{-1} \bigg|_{y=y^*},
$$
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 \( X = \{x^3_i, x^2_i, w_i^{2D} | i = 1 \cdots N\} \), 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:

\[
p(i) = \frac{\|w_i^{2D}\|_1}{\sum_{i=1}^N \|w_i^{2D}\|_1}.
\]

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 training. If the LM solver fails to find the global optimum, it is very important to the stability of Monte Carlo.

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 training. If the LM solver fails to find the global optimum, it is very important to the stability of Monte Carlo.

### 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):

\[
q_t(t) = \Gamma\left(\frac{\nu+3}{2}\right) / \Gamma\left(\frac{\nu}{2}\right) \sqrt{\nu^2 \pi^3 |\Sigma|} \left(1 + \frac{1}{\nu} \|t - \mu\|_2^2\right)^{-\frac{\nu+3}{2}}, \tag{23}
\]

where \( \|t - \mu\|_2^2 = (t - \mu)^T \Sigma^{-1} (t - \mu) \), with the location \( \mu \), the 3x3 positive definite scale matrix \( \Sigma \), and the degrees of freedom \( \nu \). Following [6], we set \( \nu \) 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 \( \Sigma_{t^*} \) is the 3x3 submatrix of the full pose covariance \( \Sigma_p^* \). Note that the actual covariance of the t-distribution is thus \( \frac{\nu}{\nu-1} \Sigma_{c^*} \), 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 \( \mu \) and scale \( \Sigma \) 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 \( \theta \), 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:

\[
q_{VM}(\theta) = \frac{\exp(\kappa \cos(\theta - \mu))}{2\pi I_0(\kappa)}, \tag{24}
\]

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

\[
q_{\text{mix}}(\theta) = (1 - \alpha)q_{VM}(\theta) + \alpha q_{\text{uniform}}(\theta), \tag{25}
\]

with the uniform mixture weight \( \alpha \). The uniform component is added in order to capture other potential modes under orientation ambiguity. We set \( \alpha \) 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 \( \theta^* \) 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 \( \mu \), we simply adopt its maximum likelihood estimation, i.e., the circular mean of the weighted samples. For the concentration \( \kappa \), we first compute an approximated estimation [7] by:

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

where \( \bar{r} = \left| \sum_j v_j |\sin \theta_j\cos \theta_j|^T / \sum_j v_j \right| \) is the norm of the mean orientation vector, with the importance weight \( v_j \) for the \( j \)-th sample \( \theta_j \). 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:

\[
q_{\text{ACG}}(l) = \frac{(I^T \Lambda^{-1} I)^{-2}}{S_4|\Lambda|^{\frac{1}{2}}}, \tag{27}
\]

where \( S_4 = 2\pi^2 \) is the 3D surface area of the 4D sphere, and \( \Lambda \) 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 \( \infty \). 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 \( t^* \) to be the PnP solution and \( \Sigma_{t^*}^{-1} \) to be the estimated 4×4 inverse covariance matrix. Note that \( \Sigma_{t^*}^{-1} \) is only valid in the local tangent space with rank 3, satisfying \( t^{*T} \Sigma_{t^*}^{-1} t^* = 0 \). The initial parameters are determined by:

\[
\Lambda \leftarrow \hat{\Lambda} + \alpha |\hat{\Lambda}|^{\frac{1}{2}} I, \tag{28}
\]

where \( \hat{\Lambda} = (\Sigma_{t^*}^{-1} + I)^{-1} \), and \( \alpha \) is a hyperparameter that controls the dispersion of the proposal for robust sampling. We set \( \alpha \) 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:

\[
\hat{\Lambda} = \frac{4}{\sum_j v_j} \sum_j v_j l_j l_j^T. \tag{29}
\]

The solution to Eq. (29) can be computed by fixed-point iteration [18]. 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 \( L_{\text{reg}} \) consists of the position loss \( L_{\text{pos}} \) and the orientation loss \( L_{\text{orient}} \).

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

\[
L_{\text{pos}} = \begin{cases} 
\frac{d_t^2}{2\beta^2}, & d_t \leq \beta, \\
\frac{1}{2} d_t - 0.5\beta, & d_t > \beta,
\end{cases} \tag{30}
\]

with the hyperparameter \( \beta \).

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

\[
L_{\text{orient}} = 1 - \cos d_\theta. \tag{31}
\]

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

\[
L_{\text{orient}} = 2 - 2((l^* + \Delta l)^T l_{gt})^2. \tag{32}
\]

For the specific settings of the hyperparameter \( \beta \) 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 \( n_{\text{head}} \) be the number of heads and \( n_{\text{hpts}} \) be the number of points per head, a total number of \( N = n_{\text{head}} n_{\text{hpts}} \) 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 \( n_{\text{head}} = 8 \), which yields 256/\( n_{\text{head}} = 32 \) 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 classifica-
tion loss for attribute. Additionally, a binary cross-entropy loss is imposed upon the 3D localization score, with the target \( c_{gt} \) defined as a function of the position error:

\[
c_{gt} = Score(\| x_{XZ}^{*} - t_{XZ,gt} \|) \\
= \max(0, \min(1, -a \log \| x_{XZ}^{*} - t_{XZ,gt} \| + b)),
\]

where \( x_{XZ}^{*} \) is the XZ components of the PnP solution, \( t_{XZ,gt} \) 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 E.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 \( x^{3D} \) and weights \( w^{3D} \) via an independent linear layer. Besides, the attention weights \( \phi \) 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:

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

where \( k \) is the head index, \( f_k(y_{gt}) \) is the weighted reprojection error of the \( k \)-th head at the truth pose \( y_{gt} \). In the above equation, the diagonal matrix \( \text{diag} w_k^{2D} \) is interpreted as the inverse square root of the covariance matrix of the normal distribution, \( i.e., \text{diag} w_k^{2D} = \Sigma^{-\frac{1}{2}} \), and the head attention weight \( \phi_k \) is interpreted as the mixture component weight. \( \sum_{RoI} \) 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 \( x^{3D} \) from the 3D LiDAR point cloud. The multi-head coordinate regression loss for each sampled point with available ground truth is defined as:

\[
L_{regr} = \sum_{k=1}^{n_{head}} \phi_k \rho \left( || x^{3D}_k - x^{3D}_{gt} ||^2 \right),
\]

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Method</th>
<th>Data</th>
<th>NDS</th>
<th>mAP</th>
<th>True positive metrics (lower is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>mATE</td>
</tr>
<tr>
<td>A0</td>
<td>Basic EPro-PnP</td>
<td>Val</td>
<td>0.425</td>
<td>0.349</td>
<td>0.676</td>
</tr>
<tr>
<td>A1</td>
<td>A0 + coord. regr.</td>
<td>Val</td>
<td>0.430</td>
<td>0.352</td>
<td>0.677</td>
</tr>
<tr>
<td>B0</td>
<td>A0 → No reprojection (L_{\text{proj}})</td>
<td>Val</td>
<td>0.408</td>
<td>0.337</td>
<td>0.721</td>
</tr>
<tr>
<td>C0</td>
<td>A0 → 50% Monte Carlo score</td>
<td>Val</td>
<td>0.424</td>
<td>0.350</td>
<td>0.673</td>
</tr>
<tr>
<td>C1</td>
<td>A0 → 100% Monte Carlo score</td>
<td>Val</td>
<td>0.424</td>
<td>0.350</td>
<td>0.675</td>
</tr>
<tr>
<td>D0</td>
<td>A1 → Compact network</td>
<td>Val</td>
<td>0.434</td>
<td>0.358</td>
<td>0.672</td>
</tr>
<tr>
<td>D1</td>
<td>D0 + TTA</td>
<td>Val</td>
<td>0.446</td>
<td>0.367</td>
<td>0.664</td>
</tr>
</tbody>
</table>

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 introduce 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 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 \(c_{\text{MC}}\) via Monte Carlo pose sampling, and compare the resulting mAP against the standard implementation with 3D score \(c_{\text{pred}}\) predicted from the object-level branch. With the PnP solution \(t^*_j\), the sampled translation vector \(t_j\), and its importance weight \(v_j\), the Monte Carlo score is computed by:

\[
c_{\text{MC}} = \frac{1}{\sum_j v_j} \sum_j v_j \text{Score}(\|t^*_j - t_{XZ,j}\|),
\]

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

\[
c_{\text{mix}} = c_{\text{MC}}^\alpha c_{\text{pred}}^{1-\alpha},
\]

where \(\alpha\) 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 $n_{npts}$ 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 1600×900 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).

<table>
<thead>
<tr>
<th>PyTorch &amp; FPN</th>
<th>Heads FCOS Deform</th>
<th>PnP</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.8.1-cu111</td>
<td>0.195 0.074 0.028</td>
<td>0.026</td>
<td>0.327</td>
</tr>
<tr>
<td>v1.10.1-cu113</td>
<td>0.172 0.056 0.025</td>
<td>0.045</td>
<td>0.301</td>
</tr>
</tbody>
</table>

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

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

Table 7. A summary of frequently used notations.
References