Supplementary Material for Projective Manifold Gradient
Layer for Deep Rotation Regression

Jiayi Chen$^{1,2}$ Yingda Yin$^1$ Tolga Birdal$^{3,4}$
$^1$CFCS, Peking University $^2$Imperial College London
$^3$Stanford University $^4$Beijing Institute for General AI

Abstract

In this supplementary material, we first provide a detailed summary of Riemannian geometry in Section 1. Subsequently, we add further details into our Riemannian optimization, justify the design of the Riemannian step size in Section 2.1, and derive the inverse projection for different rotation manifolds in Section 2.2. Since we have demonstrated an extension to the manifold of unit vectors in the paper, we also provide the gradients and optimization operators for this manifold in Section 3. We analyse the computational cost in Section 4. In addition, we report new experiment results of 3d object pose estimation from PASCAL3D+, camera relocalization on outdoor Cambridge Landscape dataset and 3d object pose estimation using flow loss in Section 5. Last but not least, we include implementation details of our experiments in Section 6 and add some information about the related rotation representation for integrity in Section 7.

1. More on Riemannian Geometry

In this part, we supplement the definitions in Section 3.1 of the main paper to allow for a slightly more rigorous specification of the exponential map for interested readers.

We denote the union of all tangent spaces as the tangent bundle: $T\mathcal{M} = \bigcup_{x \in \mathcal{M}} T_x \mathcal{M}$. Riemannian metric $G_x$ induces a norm $\|u\|_x = \sqrt{G_x(u,u)}$, $u \in T_x \mathcal{M}$ locally defining the geometry of the manifold and allows for computing the length of any curve $\gamma : [0,1] \to \mathcal{M}$, with $\gamma(0) = x$ and $\gamma(1) = y$ as the integral of its speed: $\ell(\gamma) = \int_0^1 \|\dot{\gamma}(t)\|_{\gamma(t)} dt$. The notion of length leads to a natural notion of distance by taking the infimum over all lengths of such curves, giving the Riemannian distance on $\mathcal{M}$, $d(x,y) = \inf_{\gamma} \ell(\gamma)$. The constant speed length minimizing curve $\gamma$ is called a geodesic on $\mathcal{M}$.

2. Projective Manifold Gradient on $SO(3)$

2.1. Details of Riemannian Optimization on $SO(3)$

Riemannian gradient on $SO(3)$. Since we mainly focus on the $SO(3)$ manifold in this paper, we will further show the specific expression of some related concepts of $SO(3)$ below.

Firstly, $SO(3)$ is defined as a matrix subgroup of the general linear group $GL(3)$:

$$SO(3) = \{ R \in \mathbb{R}^{3 \times 3} : R^T R = I, \det(R) = 1 \}. \quad (1)$$

The tangent space of a rotation matrix in $SO(3)$ is isomorphic to $\mathbb{R}^3$ making $SO(3)$ an embedded submanifold of the ambient Euclidean space $\mathbb{R}^3$. Hence, $SO(3)$ inherits the metric or the inner product of its embedding space, $\mathbb{R}^3$.

Since $SO(3)$ is also a Lie group, elements of the tangent space $\phi^A \in T_x \mathcal{M}$ can be uniquely mapped to the manifold $\phi$.
\[ \text{exp}_1(\phi) = I + \phi + \frac{1}{2!}(\phi^2) + \frac{1}{3!}(\phi^3) + \ldots, \] (2)

where \( I \in \text{SO}(3) \) is the identity matrix and \( \wedge \) is a skew-symmetric operator \( : \mathbb{R}^3 \to \mathbb{T}_M \) as

\[ \phi \wedge = \begin{pmatrix} 0 & -\phi_z & \phi_y \\ \phi_z & 0 & -\phi_x \\ -\phi_y & \phi_x & 0 \end{pmatrix} \] (3)

Due to the nature of the Lie group, we can expand the formula in Eq. (2) from the tangent space of the identity, \( T_H \), to \( T_H \) by simply multiplying by an \( R \):

\[ \text{exp}_R(\phi) = R \left( \sum_{n=0}^{\infty} \frac{1}{n!} (\phi^\wedge)_n \right) \] (4)

If the vector \( \phi \) is rewritten in terms of a unit vector \( \omega \) and a magnitude \( \theta \), the exponential map can further be simplified as

\[ \text{Exp}_R(\phi) = R(I + \sin \theta \, \omega \wedge + (1 - \cos \theta)(\omega \wedge)^2) \] (5)

which is well known as the Rodrigues formula [7]. Following [9], we have

\[ \frac{\partial}{\partial \phi^k} \text{Exp}_R(\phi) \bigg|_{\phi=0} = R \left( \cos \theta \, \frac{\partial }{\partial \phi^k} \omega \wedge \right) \bigg|_{\phi=0} = Rx \] (6)

where \( x = (1,0,0) \in \mathbb{R}^3 \). For \( \phi_y \) and \( \phi_z \), there are the similar expressions of the gradient. Finally, we can have

\[ \text{grad} \, \mathcal{L}(R) = \left( \frac{\partial f(R)}{\partial R} \bigg|_{\phi=0} \text{Exp}_R(\phi) \right) \wedge \] (7)

Riemannian gradient descent on \( \text{SO}(3) \). We are now ready to state the Riemannian optimization in the main paper in terms of the exponential map:

\[ R_{k+1} = \text{Exp}_R(-\tau_k \nabla \mathcal{L}). \] (8)

Note that if we consider the most commonly used L2 loss \( f(R) = ||R - R_{gt}||_F^2 \), where

\[ R = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \in \text{SO}(3), \quad R_{gt} = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{pmatrix} \in \text{SO}(3), \]

we can get an analytical expression of \( \nabla \phi_x = (\nabla \phi_x, \nabla \phi_y, \nabla \phi_z) \) as follows:

\[ \nabla \phi_x = \frac{\partial f(R)}{\partial R} \ast Rx \]

\[ = 2 \sum_{i=1}^{3} \begin{pmatrix} a_1 - x_i & b_1 - y_i & c_1 - z_i \\ a_2 - x_i & b_2 - y_i & c_2 - z_i \\ a_3 - x_i & b_3 - y_i & c_3 - z_i \end{pmatrix} \begin{pmatrix} 0 & c_1 & -b_1 \\ 0 & c_2 & -b_2 \\ 0 & c_3 & -b_3 \end{pmatrix} \]

\[ = 2 \sum_{i=1}^{3} (b_1 \ast c_1 - c_1 \ast b_1) \] (9)

\[ \nabla \phi_y = 2 * \sum_{i=1}^{3} (c_1 \ast x_i - a_1 \ast z_i) \]

\[ \nabla \phi_z = 2 * \sum_{i=1}^{3} (a_1 \ast y_i - b_1 \ast x_i). \]

\( \tau_{\text{converge}} \text{ in ablation study.} \) We have mentioned in Section 4.3 of the main paper that \( \tau \) should be small at the beginning of training and be large when converging. This is because a small \( \tau \) can yield \( R_g \) closer to \( R \) and greatly alleviate the reverse problem at the beginning stage of training discussed in Section 4.3. Later in training, a large \( \tau \) can help us converge better. The initial \( \tau \) will not influence the final results too much, and we just need to choose a reasonable value. But the final \( \tau \) matters.

Right before convergence, our ideal choice for the final \( \tau \) would be \( \tau_{gt} \). Given that the value of \( \tau_{gt} \) will change according to the geodesic distance between \( R \) and \( R_{gt} \), we instead choose to find a suitable constant value to act like \( \tau_{gt} \) when converging, which we denotes as \( \tau_{\text{converge}} \).

**Lemma 1.** The final value of \( \tau_{\text{converge}} \) satisfies:

\[ \tau_{\text{converge}} = \lim_{R,R_{gt} \to 0} R_{\text{gt}} \big( -\tau_{\text{converge}} \text{ grad } \mathcal{L}(f(R)) \big) \] (10)

where \( <R,R_{gt}> \) represents the angle between \( R \) and \( R_{gt} \).

**Proof.** Considering the symmetry, without loss of generality, we assume that \( R = I \), which will simplify the derivation. Based upon the conclusion in Eq. (9), when we use L2 loss, we have \( \nabla \phi = (2*(z_2-y_3), 2*(x_3-z_1), 2*(y_1-x_2)) \) and \( \text{grad} \, \mathcal{L}(R) = (\nabla \phi)^\wedge = 2(R_{gt} - R_{gt}) \).

Taking the manifold logarithm of both sides, we get:

\[ \log R_{\text{gt}} = \lim_{<R,R_{gt} \to 0} -\tau_{\text{converge}} \text{ grad } \mathcal{L}(f(R)) \] (11)

The solution for \( \tau_{\text{converge}} \) can then be derived as follows:

\[ \tau_{\text{converge}} = \lim_{<R,R_{gt} \to 0} -\log R_{\text{gt}} \text{ grad } \mathcal{L}(f(R)) \]

\[ = \lim_{\theta \to 0} -\frac{(\phi_{gt})^\wedge}{2(R_{gt} - R_{gt})} \]

\[ = \lim_{\theta \to 0} -\frac{\theta}{2 \sin \theta ((\omega_{gt})^\wedge)^\wedge - (\omega_{gt})^\wedge} \]

\[ = \lim_{\theta \to 0} \frac{\theta}{4 \sin \theta} \]

\[ = \frac{1}{4} \] (12)

where \( (\phi_{gt})^\wedge = \log_3 (R_{gt}) = \theta (\omega_{gt})^\wedge, \theta = <R,R_{gt}> \).

Note that though \( \tau_{\text{converge}} = \frac{1}{4} \) is only true for the L2 loss, we can solve \( \tau_{\text{converge}} \) for other frequently used loss formats, e.g., geodesic loss [5]. If we use geodesic loss \( \theta^2 \), it can be computed that \( \tau_{\text{converge}} = \frac{1}{4} \). We leave the detailed derivation to the interested readers.

**2.2. Derivations of Inverse Projection**

For different rotation representations, we follow the same process to find its inverse projection: we first find the inverse image space \( \pi^{-1}(x_{gp}) \), then project \( x \) to this space resulting in \( x_{gp} \), and finally get our (regularized) projective manifold gradient.
**Quaternion** We need to solve

$$x_{qp} = \underset{x_g \in \pi^{-1}_g(\mathbf{x})}{\arg\min} \|x_g - x\|^2_2, \quad (13)$$

where \( x \) is the raw output of our network in ambient space \( \mathbb{R}^4 \), \( \mathbf{x}_g \) is the next goal in representation manifold \( S^3 \), and \( x_g \) is the variable to optimize in ambient space \( \mathbb{R}^4 \). Recall \( \pi^{-1}_g(\mathbf{x}_g) = \{ x | x = k \mathbf{x}_g, k \in \mathbb{R} \text{ and } k > 0 \} \), and we can have

$$\|x - x_g\|^2_2 = x^2 - 2kx \cdot \mathbf{x}_g + k^2 \mathbf{x}_g^2 \quad (14)$$

Without considering the condition of \( k > 0 \), We can see when \( k = \frac{x \cdot \mathbf{x}_g}{\|x\|^2} = x \cdot \mathbf{x}_g \), the target formula reaches minimum. Note that when using a small \( \tau \), the angle between \( \mathbf{x}_g \) and \( x \) is always very small, which means the condition of \( k \cdot x \cdot \mathbf{x}_g > 0 \) can be satisfied naturally. For the sake of simplicity and consistency of gradient, we ignore the limitation of \( k \) no matter what value \( \tau \) takes. Therefore, the inverse projection is \( x_{qp} = (x \cdot \mathbf{x}_g)\mathbf{x}_g \).

**6D representation** We need to solve

$$[u_{gp}, v_{gp}] = \underset{[u_{gp}, v_{gp}] \in \pi^{-1}_g(\mathbf{u}, \mathbf{v})}{\arg\min} \left( \|u_{gp} - u\|^2_2 + \|v_{gp} - v\|^2_2 \right) \quad (15)$$

where \([u, v]\) is the raw output of network in ambient space \( \mathbb{R}^6 \), \([u_g, v_g]\) is the next goal in representation manifold \( \mathbb{V}_2(\mathbb{R}^3) \) and \([u_{gp}, v_{gp}]\) is the variable to optimize in ambient space \( \mathbb{R}^6 \). Recall \( \pi^{-1}_g([u_{gp}, v_{gp}]) = \{ [k_1 u_\theta, k_2 u_\theta + k_3 v_\theta] | k_1, k_2, k_3 \in \mathbb{R} \text{ and } k_1, k_3 > 0 \} \). We can see that \( u_g \) and \( v_g \) are independent, and \( u_g \) is similar to the situation of quaternion. So we only need to consider the part of \( v_g \) as below:

$$\|v - v_g\|^2_2 = v^2 + k_2^2 u_\theta^2 + k_3^2 v_\theta^2 - 2k_2 v \cdot u_\theta - 2k_3 v \cdot v_\theta \quad (16)$$

For the similar reason as quaternion, we ignore the condition of \( k_3 > 0 \) and we can see when \( k_2 = v \cdot u_\theta \) and \( k_3 = v \cdot v_\theta \), the target formula reaches minimum. Therefore, the inverse projection is \([u_{gp}, v_{gp}] = ([u \cdot (u \cdot \mathbf{u}_g), (v \cdot u_\theta)\mathbf{u}_g], (v \cdot v_\theta)\mathbf{v}_g]\)

**9D representation** For this representation, obtaining the inverse image \( \pi^{-1}_g \) is not so obvious. Recall \( \pi_9(D)(x) = U \Sigma V^T \), where \( U \) and \( V \) are left and right singular vectors of \( x \) decomposed by SVD expressed as \( x = U \Sigma V^T \), and \( \Sigma = \text{diag}(1, 1, \text{det}(UV^T)) \).

**Lemma 2. The inverse image** \( \pi^{-1}_g(\mathbf{R}_g) = \{ \mathbf{SR}_g | \mathbf{S} = \mathbf{S}^T \} \) satisfies that \( \{ x_g | \pi_9(D)(x_g) = \mathbf{R}_g \} \subseteq \pi^{-1}_g(\mathbf{R}_g) \).

**Proof.** To find a suitable \( \pi^{-1}_g \), the most straightforward way is to only change the singular values \( \Sigma_g = \text{diag}(\lambda_0, \lambda_1, \lambda_2) \), where \( \lambda_0, \lambda_1, \lambda_2 \) can be arbitrary scalars, and recompose the \( x_g = U \Sigma V^T \).

However, we argue that this simple method will fail to capture the entire set of \( \{ x_g | \pi_9(D)(x_g) = \mathbf{R}_g \} \), because different \( U \) and \( V \) can yield the same rotation \( \mathbf{R}_g \). In fact, \( U_g \) can be arbitrary if \( x_g = U_g \Sigma_g V_g^T \) and \( U_g \Sigma_g V_g^T = \mathbf{R}_g \). Assuming \( \mathbf{R}_g \) is known, we can replace \( V_g^T \) by \( \mathbf{R}_g \) and express \( x_g \) in a different way: \( x_g = U_g \Sigma_g \mathbf{R}_g \).

Notice that \( U_g \Sigma_g \mathbf{R}_g^{-1} \) must be a symmetry matrix since \( U_g \) is an orthogonal matrix. Therefore, \( \{ x_g | \pi_9(D)(x_g) = \mathbf{R}_g \} \subseteq \pi^{-1}_g(\mathbf{R}_g) \).

Then we need to solve

$$x_{qp} = \underset{x_g \in \pi^{-1}_g(\mathbf{R}_g)}{\arg\min} \|x_g - x\|^2_2 \quad (17)$$

where \( x \) is the raw output of our network in ambient space \( \mathbb{R}^{10 \times 3} \), \( \mathbf{x}_g \) is the next goal in representation manifold \( S^3 \), and \( x_g \) is the variable to optimize in ambient space \( \mathbb{R}^{3 \times 3} \).

We can further transform the objective function as below:

$$\|x - x_g\|^2_2 = \|\mathbf{R}_g \mathbf{S}_\mathbf{R}_g^{-1} - x\|^2_2 = \|\mathbf{S} - \mathbf{R}_g \mathbf{R}_g^{-1} \|^2_2 \quad (18)$$

Now we can easily find when \( \mathbf{S} \) equals to the symmetry part of \( \mathbf{R}_g \), the target formula reaches minimum. Therefore, the inverse projection admits a simple form \( x_{qp} = \mathbf{R}_g^{-1} + \mathbf{R}_g x \).

**10D representation** Recall the manifold mapping \( \pi_{10} : \mathbb{R}^{10} \to \mathbb{S}^3, \pi_{10}(x) \) is the next goal in representation manifold \( S^3 \), and \( x_g \) is the variable to optimize in ambient space \( \mathbb{R}^{10} \). Note that \( \lambda \) is also a variable to optimize. For the similar reason as before, for the sake of simplicity and consistency of analytical solution, here we also need to relax the constraint that \( \lambda \) should be the smallest eigenvalue of \( A(x_g) \).

To solve Eq. 19, we start from rewriting \( A(x_g)q_g = \lambda q_g \).

$$\mathbf{M} \Delta x = \lambda q_g - A(x)q_g \quad (21)$$
where $\Delta x = x_g - x$ and

$$M = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_1 & 0 & 0 & q_2 & q_3 & q_4 & 0 & 0 & 0 \\ 0 & 0 & q_1 & 0 & 0 & q_2 & q_3 & q_4 & 0 & 0 \\ 0 & 0 & 0 & q_1 & 0 & 0 & q_2 & q_3 & q_4 & 0 \\ 0 & 0 & 0 & 0 & q_1 & 0 & 0 & q_2 & q_3 & q_4 \end{pmatrix}$$

(22)

where $q_g = (q_1, q_2, q_3, q_4)^T$. For simplicity, we denote $\lambda q_g - A(x)q_g$ as $b$.

Once we have finished the above steps for preparation, we solve $\lambda$ and $\Delta x$ for the minimal problem by two steps as below. First, we assume $\lambda$ is known and the problem becomes that given $M$ and $b$, we need to find the best $\Delta x$ to minimize $\|\Delta x\|_2^2$ with the constraint $M\Delta x = b$. This is a typical quadratic optimization problem with linear equality constraints, and the analytical solution satisfies

$$\begin{pmatrix} \Delta x \\ v \end{pmatrix} = \begin{pmatrix} I & M^T \\ M & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ b \end{pmatrix}$$

(23)

where $v$ is a set of Lagrange multipliers which come out of the solution alongside $\Delta x$, and $\begin{pmatrix} I & M^T \\ M & 0 \end{pmatrix}$ is called KKT matrix. Since this matrix has full rank almost everywhere, we can multiple the inverse of this KKT matrix in both sides of Eq. 23 and lead to the solution of $\Delta x$ as below:

$$\begin{pmatrix} \Delta x \\ v \end{pmatrix} = \begin{pmatrix} I & M^T \\ M & 0 \end{pmatrix}^{-1} \begin{pmatrix} 0 \\ b \end{pmatrix}$$

(24)

Recall that $b = \lambda q_g - A(x)q_g$, therefore so far we have had the solution of $\Delta x$ respect to each $\lambda$:

$$\Delta x = \begin{pmatrix} \Delta x \\ v \end{pmatrix}_{0:10} = K(\lambda q_g - A(x)q_g) = \lambda S - T$$

(25)

in which $K$ is the upper right part of the inverse of the KKT matrix $K = \left[ \begin{pmatrix} I & M^T \\ M & 0 \end{pmatrix}^{-1} \right]_{10:14, 0:10}$, $S = Kq_g$ and $T = KA(x)q_g$.

Next, we need to optimize $\lambda$ to minimize our objective function $\|\Delta x\|_2^2$. In fact, using the results of Eq. 25, $\|\Delta x\|_2^2$ becomes a quadratic functions on $\lambda$, thus we can simply get the final analytical solution of $\lambda$ and $x_{gp}$:

$$\begin{cases} \lambda = \frac{(S^TT + T^TS)}{2S^T} \\ x_{gp} = x + \lambda S - T \end{cases}$$

(26)

Another thing worth mentioning here is that in this special case, the representation manifold $S^3$ is no longer a subspace of the ambient space $\mathbb{R}^{10}$, which means that we can’t directly compute our regularization term $x_{gp} - q_g$ because $x_{gp} \in \mathbb{R}^{10}$ while $q_g \in S^3$. However, the length vanishing problem still exists as shown in Figure 3 of our main paper. Therefore, to compute the regularization term, we need a simple mapping to convert $q_g$ to an element on $\mathbb{R}^{10}$ with stable length norm. We use the mapping $g : S^3 \rightarrow \mathbb{R}^{10}, g(q) = A^{-1}(I - qq^T)$, which is proposed in [5]. They also proved that $\pi(g(q)) = q$ is always true, which makes $g(q)$ better than simply normalizing $x_{gp}$ because the latter one will suffer from the problem of opposite gradient discussed in Section 4.3 of our main paper.

3. Projective Manifold Gradient on $S^2$

3.1. Riemannian Optimization on $S^2$

Our methods can also be applied for the regression of other manifolds. Taking $S^2$ as an example, which is included in Experiment 5.4 of our main paper, we will show the detail of how our projective manifold gradient layer works in other manifolds.

During forward, The network predicts a raw output $x \in \mathbb{R}^3$, which is then mapped to $\hat{x} \in S^2$ through a manifold mapping $\pi(x) = x/\|x\|$. Here we don’t define the rotation mapping and representation mapping, and we directly compute the loss function on representation manifold $S^2$.

During backward, to apply a Riemannian optimization, we first need to know some basic concepts of $S^2$. The tangent space of an arbitrary element $x \in S^2$ is $T_x S^2$, which is a plane. And we can map a geodesic path $v \in T_x S^2$ to an element on the manifold $S^2$ through $\exp_x(v) = \cos(||v||)x + \sin(||v||)v/||v||$, where $||.||$ means the ordinal Frobenius norm.

For the definition of the mapping $^\wedge$, which connects Euclidean space $\mathbb{R}^2$ and the tangent space $T_x S^2$, we need to first define two orthogonal axes $c_1, c_2$ in the tangent plane. Note that the choice of $c_1$ and $c_2$ won’t influence the final result, which will be shown soon after. To simplify the derivation, we can assume ground truth unit vector $x_{gt}$ is known and choose $c_1 = \frac{\log_2(x_{gt})}{\|\log_2(x_{gt})\|}x_{gt}, c_2 = \hat{x} \times c_1$. Then we can say $^\wedge = \phi_1 c_1 + \phi_2 c_2$, where $\phi = (\phi_1, \phi_2) \in \mathbb{R}^2$. The gradient of exponential mapping with respect to $\phi$ is

$$\frac{\partial}{\partial \phi_1} \exp_x(\phi)_{\phi=0} = \hat{c}_1$$

(27)

Similarly, we have

$$\frac{\partial}{\partial \phi_2} \exp_x(\phi)_{\phi=0} = \hat{c}_2$$

When using L2 loss, we can have

$$\text{grad} \, L(f) = (\nabla f(\hat{x}))^\wedge = (\nabla \phi)^\wedge$$

$$= (\frac{\partial f(\phi)}{\partial \phi} \exp_x(\phi)_{\phi=0})^\wedge$$

$$= ((2(\hat{x} - x_{gt})c_1, 2(\hat{x} - x_{gt})c_2))^\wedge$$

$$= 2((\hat{x} - x_{gt})\hat{x} - x_{gt})$$

(28)
Analytical solutions in representation mapping, inversion and projection steps. RPMG also has a very marginal cost on the memory, as it does not introduce any weights but only a few intermediate variables.

5. More Experiments

5.1. Pascal3D+

Pascal3D+ [12] is a standard benchmark for object pose estimation from real images. We follow the same setting as in [4] to estimate object poses from single images. For training we discard occluded or truncated objects and augment with rendered images from [8]. In the Table 1 and Table 2, we report our results on sofa and bicycle categories. We use the same batch size as in [4]. As for the learning rate, we use the same strategy as in Experiment 5.2 of our main paper. See the discussion in Section 6.2.

It can be seen that our method leads to consistent improvements to quaternion, 6D, 9D and 10D representations on both sofa and bicycle classes. One may be curious about why our method can only outperform 9D-inf for a margin. We think that this is because this dataset is quite challenging, the number of annotated real image for training is only around 200 for each category. Though there are a lot of synthetic images generated from [8] for training, these images suffer from sim-to-real domain gap. Therefore, we argue that the bottleneck here is not in optimization, which makes the gains from less noise in gradient smaller (Note that 9D-inf
5.2. Camera Relocalization

The task of camera relocalization is to estimate a 6 Degree-of-Freedom camera pose (rotation and translation) from visual observations, which is a fundamental component of many computer vision and robotic applications. In this experiment, we use all the settings (data, network, training strategy, hyperparameters, etc.) of PoseLSTM [10] except that we modify the rotation representations and the gradient layers. We report the results on the outdoor Cambridge Landscape dataset [3] in Table 3.

Notice that our RPMG layer performs the best on the rotation regression task, but not on the translation regression. We believe this results from a loss imbalance. We does not change the weights of the rotation loss and translation loss, otherwise it leads to an unfair comparison with existing results. We only care about the rotation error here.

5.3. Using Flow Loss for Rotation Estimation from Point Clouds.

Apart from the most widely used L2 loss, our method can also be applied to the loss of other forms, e.g. flow loss. We mainly follow the setting in Experiment 5.1 of our main paper with airplane point clouds dataset and the only difference is that we use flow loss $\|R_X - R_{gt}X\|^2_F$ here, where $X$ is the complete point cloud.

Since the format of loss is changed, the previous schedule of $\tau$ is not suitable anymore, and we have to change the value of $\tau$ accordingly. Our selection skill is to first choose a $\tau$ as we like and visualize the mean geodesic distance between predicted $R$ and $R_{gt}$ during training. Then we can roughly adjust $\tau$ to make the geodesic distance looked reasonable. For this experiment, we use $\tau = 50$ and $\lambda = 0.01$. In Table 4, we show our methods again outperform vanilla methods as well as 9D-inf.

6. More Implementation Details

6.1. Experiment 5.1 & 5.3 & 5.4 of Main Paper

Data We generate the data from ModelNet dataset [11] by sampling 1024 points on the mesh surface, following the same generation method as in [13]. We uniformly sample M rotations for each data point and set them as the ground truth. We apply the sampled rotations on the canonical point clouds to obtain the input data.

Network Architecture We use a PointNet++ MSG [6] backbone as our feature extractor. Our network takes input a point cloud with a resolution of 1024. It then performs three set abstractions to lower the resolution to 512, 128, and finally 1, resulting in a global feature of dimensionality 1024. The feature is finally pushed through a three-layer...
MLP \([1024, 512, N]\) to regress rotation, where \(N\) is the dimension of the rotation representation.

**Training details** The learning rate is set to 1e-3 and decayed by 0.7 every 3k iterations. The batch size is 20. For each experiment, we train the network on one NVIDIA TITAN Xp GPU for 30k iterations.

### 6.2. Experiment 5.2 of Main Paper

Most of the training settings and strategies are all the same as [4] except learning rate. We find setting initial learning rate \(lr = 1e-3\) and decaying to \(1e-5\) can perform much better than using \(lr = 1e-5\) as in [4], which accounts for the inconsistency of the results of those baseline methods compared to [4]. We believe that the methods should be compared under hyperparameters as optimal as possible. Thus, we stick to our \(lr\) schedule.

![Image](image.png)

Table 5. Test Error Percentiles for Experiment 5.1&5.2 in our main paper. Left: test error percentiles of airplane for Experiment 5.1 after training completes. Right: test error percentiles of chair for Experiment 5.2 after training completes.

### 7. Addition on Rotation Representations

**Standard mapping between rotation matrix and unit quaternion** The rotation mapping \(\phi : q \mapsto R\) algebraically manipulates a unit quaternion \(q\) into a rotation matrix:

\[
\phi(q) = \begin{pmatrix}
2(q_0^2 + q_1^2) - 1 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\
2(q_1q_2 + q_0q_3) & 2(q_2^2 + q_3^2) - 1 & 2(q_2q_3 - q_1q_0) \\
2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_1q_0) & 2(q_3^2 + q_0^2) - 1
\end{pmatrix}
\]

(30)

where \(q = (q_0, q_1, q_2, q_3) \in S^3\).

In the reverse direction, the representation mapping \(\psi(R)\) can be expressed as:

\[
\begin{align*}
q_0 &= \sqrt{1 + R_{00} + R_{11} + R_{22}}/2 \\
q_1 &= (R_{21} - R_{12})/(4 \times q_0) \\
q_2 &= (R_{02} - R_{20})/(4 \times q_0) \\
q_3 &= (R_{10} - R_{01})/(4 \times q_0)
\end{align*}
\]

(31)

Note that \(q = (q_0, q_1, q_2, q_3)\) and \(-q = (-q_0, -q_1, -q_2, -q_3)\) both are the valid quaternions parameterizing the same \(R\).

### References


