Supplementary Material for Projective Manifold Gradient Layer for Deep Rotation Regression

Jiayi Chen[1,2] Yingda Yin[1] Tolga Birdal[3,4] Baoquan Chen[1] Leonidas J. Guibas[3] He Wang[1†]
[1]CFCS, Peking University [2]Beijing Institute for General AI
[3]Stanford University [4]Imperial College London

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*: $\mathcal{TM} = \cup_{\mathbf{x}\in\mathcal{M}}\mathcal{T}_\mathbf{x}\mathcal{M}$. Riemannian metric $\mathbf{G}_\mathbf{x}$ induces a norm $\|\mathbf{u}\|_\mathbf{x}, \forall \mathbf{u} \in \mathcal{T}_\mathbf{x}\mathcal{M}$ locally defining the geometry of the manifold and allows for computing the *length* of any curve $\gamma : [0, 1] \to \mathcal{M}$, with $\gamma(0) = \mathbf{x}$ and $\gamma(1) = \mathbf{y}$ 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 $\mathcal{M}$, $d(\mathbf{x}, \mathbf{y}) = \inf_\gamma \ell(\gamma)$. The constant speed *length minimizing* curve γ is called a *geodesic* on $\mathcal{M}$.

By the celebrated Picard Lindelöf theorem [2], given any $(\mathbf{x}, \mathbf{v}) \in \mathcal{TM}$, there exists a unique *maximal* geodesic $\gamma_\mathbf{v}$ such that $\gamma_\mathbf{v}(0) = \mathbf{x}$ and $\dot\gamma_\mathbf{v}(0) = \mathbf{v}$. Hence, we can define a unique diffeomorphism or *exponential map*, sending $\mathbf{x}$ to the endpoint of the geodesic: $\exp_\mathbf{x}(\mathbf{v}) = \gamma_\mathbf{v}(1)$. We will refer to the well-defined, smooth inverse of this map as the *logaritmic map*: $\log_\mathbf{x}\mathbf{y} \triangleq \exp_\mathbf{x}^{-1}(\mathbf{v})$. Note that the geodesic is not the only way to move away from $\mathbf{x}$ in the direction of $\mathbf{v}$ on $\mathcal{M}$. In fact, any continuously differentiable, smooth map $R_\mathbf{x} : \mathcal{T}_\mathbf{x}\mathcal{M} \mapsto \mathcal{M}$ whose directional derivative along $\mathbf{v}$ is identity, *i.e.* $\mathrm{D}R_\mathbf{x}(\mathbf{0})[\mathbf{v}] = \mathbf{v}$ and $R_\mathbf{x}(\mathbf{0}) = \mathbf{x}$ allows for moving on the manifold in a given direction $\mathbf{v}$. Such $R_\mathbf{x}$, called *retraction*, constitutes the basic building block of any on-manifold optimizer as we use in the main paper. In addition to those we also speak of a *manifold projector* $\pi : \mathcal{X} \mapsto \mathcal{M}$ is available for the manifolds we consider in this paper. Note that, most of these definitions directly generalize to matrix manifolds such as Stiefel or Grassmann [1].

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 $GL(3)$:

$$\mathrm{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times3} : \mathbf{R}^\top\mathbf{R} = \mathbf{I}, \det(\mathbf{R}) = 1\}. \quad (1)$$

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 Eucldiean space $\mathcal{X}$. Hence, SO(3) *inherits* the metric or the inner product of its embedding space, $\mathcal{X}$.

Since SO(3) is also a Lie group, elements of the tangent space $\boldsymbol{\phi}^\wedge \in \mathcal{T}_\mathbf{I}\mathcal{M}$ can be uniquely mapped to the manifold

[†]: He Wang is the corresponding author (hewang@pku.edu.cn).

maximal refers to the fact that the curve is as long as possible.

$\mathcal{M}$ through the exponential map:

$$\exp_{\mathbf{I}}(\boldsymbol{\phi}^{\wedge}) = \mathbf{I} + \boldsymbol{\phi}^{\wedge} + \frac{1}{2!}(\boldsymbol{\phi}^{\wedge})^2 + \frac{1}{3!}(\boldsymbol{\phi}^{\wedge})^3 + \dots \quad , \tag{2}$$

where $\mathbf{I} \in \mathrm{SO}(3)$ is the identity matrix and $^\wedge$ is a skew-symmetric operator $^\wedge : \mathbb{R}^3 \to \mathcal{T}_{\mathbf{I}}\mathcal{M}$ as

$$\boldsymbol{\phi}^{\wedge} = \begin{pmatrix} 0 & -\phi_z & \phi_y \\ \phi_z & 0 & -\phi_x \\ -\phi_y & \phi_x & 0 \end{pmatrix} \tag{3}$$

Due to the nature of the Lie group, we can expand the formula in Eq. (2) from the tangent space of the identity, $\mathcal{T}_{\mathbf{I}}\mathcal{M}$, to $\mathcal{T}_{\mathbf{R}}\mathcal{M}$ by simply multiplying by an $\mathbf{R}$:

$$\exp_{\mathbf{R}}(\boldsymbol{\phi}^{\wedge}) = \mathbf{R}\left(\sum_{n=0}^{\infty}(\frac{1}{n!}(\boldsymbol{\phi}^{\wedge})^n)\right) \tag{4}$$

If the vector $\boldsymbol{\phi}$ is rewritten in terms of a unit vector $\boldsymbol{\omega}$ and a magnitude θ, the exponential map can further be simplified as

$$\mathrm{Exp}_{\mathbf{R}}(\boldsymbol{\phi}) = \mathbf{R}(\mathbf{I} + \sin\theta\,\boldsymbol{\omega}^{\wedge} + (1-\cos\theta)(\boldsymbol{\omega}^{\wedge})^2) \tag{5}$$

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

$$\frac{\partial}{\partial \phi_x}\mathrm{Exp}_{\mathbf{R}}(\boldsymbol{\phi})\bigg|_{\boldsymbol{\phi}=\mathbf{0}} = \mathbf{R}\left(\cos\theta\,\frac{\partial\theta}{\partial\phi_x}\boldsymbol{\omega}^{\wedge}\right)\bigg|_{\boldsymbol{\phi}=\mathbf{0}} = \mathbf{R}\mathbf{x}^{\wedge} \tag{6}$$

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

$$\mathrm{grad}\,\mathcal{L}f(\mathbf{R}) = \left(\frac{\partial f(\mathbf{R})}{\partial \mathbf{R}}\frac{\partial}{\partial \boldsymbol{\phi}}\mathrm{Exp}_{\mathbf{R}}(\boldsymbol{\phi})\bigg|_{\boldsymbol{\phi}=\mathbf{0}}\right)^{\wedge} \tag{7}$$

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

$$\mathbf{R}_{k+1} = \mathrm{Exp}_{\mathbf{R}_k}(-\tau_k\nabla\boldsymbol{\phi}). \tag{8}$$

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

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

we can get an analytical expression of $\nabla\boldsymbol{\phi} = (\nabla\phi_x, \nabla\phi_y, \nabla\phi_z)$ as follows:

$$\begin{aligned} \nabla\boldsymbol{\phi}_x &= \frac{\partial f(\mathbf{R})}{\partial \mathbf{R}} * \mathbf{R}\mathbf{x}^{\wedge} \\ &= 2\left\|\begin{pmatrix} a_1-x_1 & b_1-y_1 & c_1-z_1 \\ a_2-x_2 & b_2-y_2 & c_2-z_2 \\ a_3-x_3 & b_3-y_3 & c_3-z_3 \end{pmatrix}\begin{pmatrix} 0 & c_1 & -b_1 \\ 0 & c_2 & -b_2 \\ 0 & c_3 & -b_3 \end{pmatrix}\right\|_1 \\ &= 2 * \sum_{i=1}^{3}(b_i * z_i - c_i * y_i) \end{aligned} \tag{9}$$

Similarly, we have $\nabla\phi_y = 2 * \sum_{i=1}^{3}(c_i * x_i - a_i * z_i)$ and $\nabla\phi_z = 2 * \sum_{i=1}^{3}(a_i * y_i - b_i * x_i)$.

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

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

Lemma 1. *The final value of $\tau_{converge}$ satisfies:*

$$\mathbf{R}_{\mathrm{gt}} = \lim_{<\mathbf{R},\mathbf{R}_{\mathrm{gt}}>\to 0} R_{\mathbf{R}}(-\tau_{converge}\,\mathrm{grad}\,\mathcal{L}(f(\mathbf{R}))) \tag{10}$$

where $<\mathbf{R},\mathbf{R}_{\mathrm{gt}}>$ *represents the angle between* $\mathbf{R}$ *and* $\mathbf{R}_{\mathrm{gt}}$.

Proof. Considering the symmetry, without loss of generality, we assume that $\mathbf{R} = \mathbf{I}$, which will simplify the derivation. Based upon the conclusion in Eq. (9), when we use L2 loss, we have $\nabla\boldsymbol{\phi} = (2*(z_2-y_3), 2*(x_3-z_1), 2*(y_1-x_2))$ and $\mathrm{grad}\,\mathcal{L}f(\mathbf{R}) = (\nabla\boldsymbol{\phi})^{\wedge} = 2(\mathbf{R}_{\mathrm{gt}}^{\top} - \mathbf{R}_{\mathrm{gt}})$. Taking the manifold logarithm of both sides, we get:

$$\log_{\mathbf{R}}(\mathbf{R}_{\mathrm{gt}}) = \lim_{<\mathbf{R},\mathbf{R}_{\mathrm{gt}}>\to 0} -\tau_{converge}\,\mathrm{grad}\,\mathcal{L}f(\mathbf{R}) \tag{11}$$

The solution for $\tau_{converge}$ can then be derived as follows:

$$\begin{aligned} \tau_{converge} &= \lim_{<\mathbf{R},\mathbf{R}_{\mathrm{gt}}>\to 0} -\frac{\log_{\mathbf{R}}(\mathbf{R}_{\mathrm{gt}})}{\mathrm{grad}\,\mathcal{L}f(\mathbf{R})} \\ &= \lim_{\theta\to 0} -\frac{(\boldsymbol{\phi}_{\mathrm{gt}})^{\wedge}}{2(\mathbf{R}_{\mathrm{gt}}^{\top} - \mathbf{R}_{\mathrm{gt}})} \\ &= \lim_{\theta\to 0} -\frac{(\boldsymbol{\phi}_{\mathrm{gt}})^{\wedge}}{2\sin\theta(((\boldsymbol{\omega}_{\mathrm{gt}})^{\wedge})^{\top} - (\boldsymbol{\omega}_{\mathrm{gt}})^{\wedge})} \\ &= \lim_{\theta\to 0}\frac{\theta}{4\sin\theta} \\ &= \frac{1}{4} \end{aligned} \tag{12}$$

where $(\boldsymbol{\phi}_{\mathrm{gt}})^{\wedge} = \log_{\mathbf{I}}(\mathbf{R}_{\mathrm{gt}}) = \theta(\boldsymbol{\omega}_{\mathrm{gt}})^{\wedge}, \theta =<\mathbf{I},\mathbf{R}_{\mathrm{gt}}>$ □

Note that though $\tau_{converge} = \frac{1}{4}$ is only true for the L2 loss, we can solve $\tau_{converge}$ for other frequently used loss formats, *e.g.*, geodesic loss [5]. If we use geodesic loss θ^2, it can be computed that $\tau_{converge} = \frac{1}{2}$. 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 $\pi^{-1}(\mathbf{x}_g)$, then project $\mathbf{x}$ to this space resulting in $\mathbf{x}_{gp}$, and finally get our (regularized) projective manifold gradient.

Quaternion We need to solve

$$\mathbf{x}_{gp} = \operatorname*{argmin}_{\mathbf{x}_g \in \pi_q^{-1}(\hat{\mathbf{x}}_g)} \|\mathbf{x}_g - \mathbf{x}\|_2^2, \tag{13}$$

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

$$\|\mathbf{x} - \mathbf{x}_g\|_2^2 = \mathbf{x}^2 - 2k\mathbf{x} \cdot \hat{\mathbf{x}}_g + k^2\hat{\mathbf{x}}_g^2 \tag{14}$$

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

6D representation We need to solve

$$[\mathbf{u}_{gp}, \mathbf{v}_{gp}] = \operatorname*{argmin}_{[\mathbf{u}_g, \mathbf{v}_g] \in \pi_{6D}^{-1}([\hat{\mathbf{u}}_g, \hat{\mathbf{v}}_g])} (\|\mathbf{u}_g - \mathbf{u}\|_2^2 + \|\mathbf{v}_g - \mathbf{v}\|_2^2) \tag{15}$$

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

$$\|\mathbf{v} - \mathbf{v}_g\|_2^2 = \mathbf{v}^2 + k_2^2\hat{\mathbf{u}}_g^2 + k_3^2\hat{\mathbf{v}}_g^2 - 2k_2\mathbf{v}\cdot\hat{\mathbf{u}}_g - 2k_3\mathbf{v}\cdot\hat{\mathbf{v}}_g \tag{16}$$

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

9D representation For this representation, obtaining the inverse image π_{9D}^{-1} is not so obvious. Recall $\pi_{9D}(\mathbf{x}) = \mathbf{U}\Sigma'\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ are left and right singular vectors of $\mathbf{x}$ decomposed by SVD expressed as $\mathbf{x} = \mathbf{U}\Sigma\mathbf{V}^\top$, and $\Sigma' = \operatorname{diag}(1, 1, \det(\mathbf{U}\mathbf{V}^\top))$.

Lemma 2. *The inverse image* $\pi_{9D}^{-1}(\mathbf{R}_g) = \{\mathbf{S}\mathbf{R}_g \mid \mathbf{S} = \mathbf{S}^\top\}$ *satisfies that* $\{\mathbf{x}_g \mid \pi_{9D}(\mathbf{x}_g) = \mathbf{R}_g\} \subset \pi_{9D}^{-1}(\mathbf{R}_g)$.

Proof. To find a suitable π_{9D}^{-1}, the most straightforward way is to only change the singular values $\Sigma_g = \operatorname{diag}(\lambda_0, \lambda_1, \lambda_2)$, where $\lambda_0, \lambda_1, \lambda_2$ can be arbitrary scalars, and recompose the $\mathbf{x}_g = \mathbf{U}\Sigma_g\mathbf{V}^\top$.

However, we argue that this simple method will fail to capture the entire set of $\{\mathbf{x}_g \mid \pi_{9D}(\mathbf{x}_g) = \mathbf{R}_g\}$, because different $\mathbf{U}'$ and $\mathbf{V}'$ can yield the same rotation $\mathbf{R}_g$. In fact, $\mathbf{U}_g$ can be arbitrary if $\mathbf{x}_g = \mathbf{U}_g\Sigma_g\mathbf{V}_g^\top$ and $\mathbf{U}_g\Sigma_g'\mathbf{V}_g^\top = \mathbf{R}_g$. Assuming $\mathbf{R}_g$ is known, we can replace $\mathbf{V}_g^\top$ by $\mathbf{R}_g$ and express $\mathbf{x}_g$ in a different way: $\mathbf{x}_g = \mathbf{U}_g\Sigma_g\frac{1}{\Sigma_g'}\mathbf{U}_g^{-1}\mathbf{R}_g$. Notice that $\mathbf{U}_g\Sigma_g\frac{1}{\Sigma_g'}\mathbf{U}_g^{-1}$ must be a symmetry matrix since $\mathbf{U}_g$ is an orthogonal matrix. Therefore, $\{\mathbf{x}_g \mid \pi_{9D}(\mathbf{x}_g) = \mathbf{R}_g\} \subseteq \pi_{9D}^{-1}(\mathbf{R}_g) = \{\mathbf{S}\mathbf{R}_g \mid \mathbf{S} = \mathbf{S}^\top\}$.

Note that such $\mathbf{x}_g \in \pi_{9D}^{-1}(\mathbf{R}_g)$ can't ensure $\pi_{9D}(\mathbf{x}_g) = \mathbf{R}_g$, because in the implementation of SVD, the order and the sign of three singular values are constrained, which is not taken into consideration. Therefore, $\{\mathbf{x}_g \mid \pi_{9D}(\mathbf{x}_g) = \mathbf{R}_g\} \neq \pi_{9D}^{-1}(\mathbf{R}_g)$. □

Then we need to solve

$$\mathbf{x}_{gp} = \operatorname*{argmin}_{\mathbf{x}_g \in \pi_{9D}^{-1}(\mathbf{R}_g)} \|\mathbf{x}_g - \mathbf{x}\|_2^2 \tag{17}$$

where $\mathbf{x}$ is the raw output of our network in *ambient space* $\mathbb{R}^{3\times3}$, $\hat{\mathbf{x}}_g$ is the next goal in *representation manifold* $\mathrm{SO}(3)$, and $\mathbf{x}_g$ is the variable to optimize in *ambient space* $\mathbb{R}^{3\times3}$. We can further transform the objective function as below:

$$\|\mathbf{x}_g - \mathbf{x}\|_2^2 = \|\mathbf{S}\mathbf{R}_g - \mathbf{x}\|_2^2 = \|\mathbf{S} - \mathbf{x}\mathbf{R}_g^\top\|_2^2 \tag{18}$$

Now we can easily find when $\mathbf{S}$ equals to the symmetry part of $\mathbf{x}\mathbf{R}_g^\top$, the target formula reaches minimum. Therefore, the inverse projection admits a simple form $\mathbf{x}_{gp} = \frac{\mathbf{x}\mathbf{R}_g^\top + \mathbf{R}_g\mathbf{x}^\top}{2}\mathbf{R}_g$.

10D representation Recall the *manifold mapping* $\pi_{10D} : \mathbb{R}^{10} \to \mathcal{S}^3, \pi_{10D}(\mathbf{x}) = \min_{\mathbf{q}\in\mathcal{S}^3} \mathbf{q}^\top\mathbf{A}(\mathbf{x})\mathbf{q}$, in which

$$\mathbf{A}(\boldsymbol{\theta}) = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 & \theta_4 \\ \theta_2 & \theta_5 & \theta_6 & \theta_7 \\ \theta_3 & \theta_6 & \theta_8 & \theta_9 \\ \theta_4 & \theta_7 & \theta_9 & \theta_{10} \end{pmatrix}. \tag{19}$$

We need to solve

$$\mathbf{x}_{gp} = \operatorname*{arg\,min}_{\mathbf{A}(\mathbf{x}_g)\mathbf{q}_g = \lambda\mathbf{q}_g} \|\mathbf{x}_g - \mathbf{x}\|_2^2, \tag{20}$$

where $\mathbf{x}$ is the raw output of our network in *ambient space* $\mathbb{R}^{10}$, $\mathbf{q}_g$ is the next goal in *representation manifold* $\mathcal{S}^3$, and $\mathbf{x}_g$ is the variable to optimize in *ambient space* $\mathbb{R}^{10}$. Note that λ 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 λ should be the smallest eigenvalue of $\mathbf{A}(\mathbf{x}_g)$.

To solve Eq. 19, we start from rewriting $\mathbf{A}(\mathbf{x}_g)\mathbf{q}_g = \lambda\mathbf{q}_g$ as

$$\mathbf{M}\Delta\mathbf{x} = \lambda\mathbf{q}_g - \mathbf{A}(\mathbf{x})\mathbf{q}_g, \tag{21}$$

where $\Delta\mathbf{x} = \mathbf{x}_g - \mathbf{x}$ and

$$\mathbf{M} = \begin{pmatrix} q_1 & q_2 & q_3 & q_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_1 & 0 & 0 & q_2 & q_3 & q_4 & 0 & 0 & 0 \\ 0 & 0 & q_1 & 0 & 0 & q_2 & 0 & q_3 & q_4 & 0 \\ 0 & 0 & 0 & q_1 & 0 & 0 & q_2 & 0 & q_3 & q_4 \end{pmatrix} \quad (22)$$

where $\mathbf{q}_g = (q_1, q_2, q_3, q_4)^\top$. For simplicity, we denote $\lambda\mathbf{q}_g - \mathbf{A}(\mathbf{x})\mathbf{q}_g$ as $\mathbf{b}$.

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

$$\begin{pmatrix} \mathbf{I} & \mathbf{M}^\top \\ \mathbf{M} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \Delta\mathbf{x} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix} \quad (23)$$

where $\mathbf{v}$ is a set of Lagrange multipliers which come out of the solution alongside $\Delta\mathbf{x}$, and $\begin{pmatrix} \mathbf{I} & \mathbf{M}^\top \\ \mathbf{M} & \mathbf{0} \end{pmatrix}$ 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 $\Delta\mathbf{x}$ as below:

$$\begin{pmatrix} \Delta\mathbf{x} \\ \mathbf{v} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{M}^\top \\ \mathbf{M} & \mathbf{0} \end{pmatrix}^{-1} \begin{pmatrix} \mathbf{0} \\ \mathbf{b} \end{pmatrix} \quad (24)$$

Recall that $\mathbf{b} = \lambda\mathbf{q}_g - \mathbf{A}(\mathbf{x})\mathbf{q}_g$, therefore so far we have had the solution of $\Delta\mathbf{x}$ respect to each λ:

$$\Delta\mathbf{x} = \begin{pmatrix} \Delta\mathbf{x} \\ \mathbf{v} \end{pmatrix}_{0:10} = \mathbf{K}(\lambda\mathbf{q}_g - \mathbf{A}(\mathbf{x})\mathbf{q}_g) = \lambda\mathbf{S} - \mathbf{T} \quad (25)$$

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

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

$$\begin{cases} \lambda = \frac{(\mathbf{S}^\top\mathbf{T} + \mathbf{T}^\top\mathbf{S})}{2\mathbf{S}^\top\mathbf{S}} \\ \mathbf{x}_{gp} = \mathbf{x} + \lambda\mathbf{S} - \mathbf{T} \end{cases} \quad (26)$$

Another thing worth mentioning here is that in this special case, the *representation manifold* $\mathcal{S}^3$ is no longer a subspace of the *abmient space* $\mathbb{R}^{10}$, which means that we can't directly compute our regularization term $\mathbf{x}_{gp} - \mathbf{q}_g$ because $\mathbf{x}_{gp} \in \mathbb{R}^{10}$ while $\mathbf{q}_g \in \mathcal{S}^3$. 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 $\mathbf{q}_g$ to an element on $\mathbb{R}^{10}$ with stable length norm. We use the mapping $g: \mathcal{S}^3 \to \mathbb{R}^{10}, g(\mathbf{q}) = \mathbf{A}^{-1}(\mathbf{I} - \mathbf{q}\mathbf{q}^\top)$, which is proposed in [5]. They also proved that $\pi(g(\mathbf{q})) = \mathbf{q}$ is always true, which makes $g(\mathbf{q})$ better than simply normalizing $\mathbf{x}_{gp}$ 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 $\mathcal{S}^2$ 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 $\mathbf{x} \in \mathbb{R}^3$, which is then mapped to $\hat{\mathbf{x}} \in \mathcal{S}^2$ through a *manifold mapping* $\pi(\mathbf{x}) = \mathbf{x}/\|\mathbf{x}\|$. Here we don't define the *rotation mapping* and *representation mapping*, and we directly compute the loss function on *representation manifold* $\mathcal{S}^2$.

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

For the definition of the mapping $^\wedge$, which connects Euclidean space $\mathbb{R}^2$ and the tangent space $\mathcal{T}_{\hat{\mathbf{x}}}\mathcal{M}$, we need to first define two orthogonal axes $\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2$ in the tangent plane. Note that the choice of $\hat{\mathbf{c}}_1$ and $\hat{\mathbf{c}}_2$ won't influence the final result, which will be shown soon after. To simplify the derivation, we can assume ground truth unit vector $\hat{\mathbf{x}}_{gt}$ is known and choose $\hat{\mathbf{c}}_1 = \frac{\mathrm{Log}_{\hat{\mathbf{x}}}(\hat{\mathbf{x}}_{gt})}{\|\mathrm{Log}_{\hat{\mathbf{x}}}(\hat{\mathbf{x}}_{gt})\|} = \frac{\hat{\mathbf{x}}_{gt} - (\hat{\mathbf{x}}_{gt}\cdot\hat{\mathbf{x}})}{\|\hat{\mathbf{x}}_{gt} - (\hat{\mathbf{x}}_{gt}\cdot\hat{\mathbf{x}})\|}$ and $\hat{\mathbf{c}}_2 = \hat{\mathbf{x}} \times \hat{\mathbf{c}}_1$. Then we can say $\boldsymbol{\phi}^\wedge = \phi_1\hat{\mathbf{c}}_1 + \phi_2\hat{\mathbf{c}}_2$, where $\boldsymbol{\phi} = (\phi_1, \phi_2) \in \mathbb{R}^2$. The gradient of exponential mapping with respect to $\boldsymbol{\phi}$ is

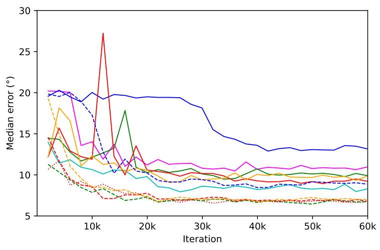
$$\begin{aligned} &\frac{\partial}{\partial\phi_1}\mathrm{Exp}_{\hat{\mathbf{x}}}(\boldsymbol{\phi})\Big|_{\boldsymbol{\phi}=\mathbf{0}} \\ &= \frac{\partial}{\partial\phi_1}(\cos(\|\phi_1\hat{\mathbf{c}}_1\|)\hat{\mathbf{x}} + \sin(\|\phi_1\hat{\mathbf{c}}_1\|)\frac{\phi_1\hat{\mathbf{c}}_1}{\|\phi_1\hat{\mathbf{c}}_1\|})\Big|_{\boldsymbol{\phi}=\mathbf{0}} \\ &= \hat{\mathbf{c}}_1 \end{aligned} \quad (27)$$

Similarly, we have $\frac{\partial}{\partial\phi_2}\mathrm{Exp}_{\hat{\mathbf{x}}}(\boldsymbol{\phi})\Big|_{\boldsymbol{\phi}=\mathbf{0}} = \hat{\mathbf{c}}_2$.

When using L2 loss, we can have

$$\begin{aligned} \mathrm{grad}\,\mathcal{L}f(\hat{\mathbf{x}}) &= (\nabla f(\hat{\mathbf{x}}))^\wedge = (\nabla\boldsymbol{\phi})^\wedge \\ &= \left(\frac{\partial f(\hat{\mathbf{x}})}{\partial\hat{\mathbf{x}}}\frac{\partial}{\partial\boldsymbol{\phi}}\mathrm{Exp}_{\hat{\mathbf{x}}}(\boldsymbol{\phi})\Big|_{\boldsymbol{\phi}=\mathbf{0}}\right)^\wedge \\ &= ((2(\hat{\mathbf{x}} - \hat{\mathbf{x}}_{gt})\hat{\mathbf{c}}_1, 2(\hat{\mathbf{x}} - \hat{\mathbf{x}}_{gt})\hat{\mathbf{c}}_2))^\wedge \\ &= 2((\hat{\mathbf{x}}\cdot\hat{\mathbf{x}}_{gt})\hat{\mathbf{x}} - \hat{\mathbf{x}}_{gt}) \end{aligned} \quad (28)$$

Methods	Accuracy(%) ↑			Med(°) ↓
	10°	15°	20°	Err
Euler	60.2	80.9	90.6	8.3
Axis-Angle	45.0	70.9	85.1	11.0
Quaternion	34.3	60.8	73.5	13.2
6D	50.8	76.7	89.0	9.9
9D	52.4	79.6	90.3	9.2
9D-Inf	70.9	**88.0**	93.5	**6.7**
10D	50.2	77.0	89.6	9.8
RPMG-Quat	56.6	79.6	90.9	8.9
RPMG-6D	69.6	86.1	92.2	**6.7**
RPMG-9D	**72.5**	**88.0**	**95.8**	**6.7**
RPMG-10D	69.3	87.1	93.9	7.0



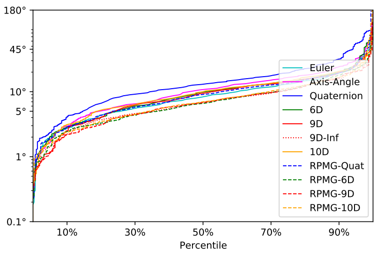
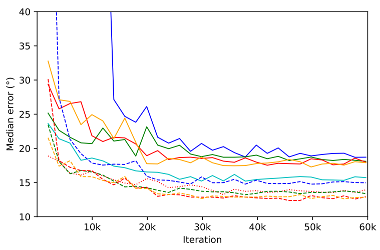


Table 1. **Pose estimation from PASCAL3D+ *sofa* images.** Left: a comparison of methods by 10° / 15° / 20° accuracy of (geodesic) errors and median errors after 60k training steps. Middle: median test error at different iterations during training. Right: test error percentiles after training completes. The legend on the right applies to both plots.

Methods	Accuracy(%) ↑			Med(°) ↓
	10°	15°	20°	Err
Euler	28.2	48.1	62.7	15.7
Axis-Angle	5.3	8.1	10.1	79.7
Quaternion	20.8	38.8	54.6	18.7
6D	21.8	39.0	55.3	18.1
9D	20.6	37.6	56.9	18.0
9D-Inf	38.0	53.3	69.9	13.4
10D	23.9	42.3	56.7	17.9
RPMG-Quat	32.3	50.0	65.6	15.0
RPMG-6D	35.4	57.2	70.6	13.5
RPMG-9D	36.8	57.4	**71.8**	**12.5**
RPMG-10D	**40.0**	**57.7**	71.3	12.9



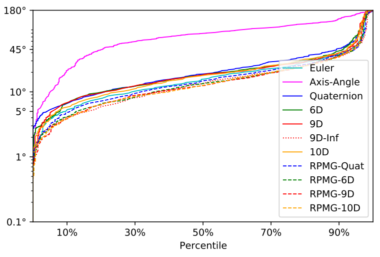


Table 2. **Pose estimation from PASCAL3D+ *bicycle* images.** We report the same metrics as Table 1; see the caption there.

Note that this expression doesn't depend on the choice of $\hat{\mathbf{c}}_1$ and $\hat{\mathbf{c}}_2$.

Similar to Eq 12, we can also solve a $\tau_{converge}$

$$\tau_{converge} = \lim_{<\hat{\mathbf{x}}, \hat{\mathbf{x}}_{\text{gt}}> \to 0} -\frac{\text{Log}_{\hat{\mathbf{x}}}(\hat{\mathbf{x}}_{\text{gt}})}{\text{grad}\ \mathcal{L}f(\hat{\mathbf{x}})} = \lim_{\theta \to 0} \frac{\theta \hat{\mathbf{c}}_1}{2 \sin \theta \hat{\mathbf{c}}_1} = \frac{1}{2} \tag{29}$$

where $\theta = < \hat{\mathbf{x}}, \hat{\mathbf{x}}_{gt} >$. Note that in Experiment 5.4, we change the schedule of τ according to this conclusion. We increase τ from 0.1 to 0.5 by uniform steps.

3.2. Inverse Projection

Similar to quaternion, we can have $\mathbf{x}_{gp} = (\mathbf{x} \cdot \hat{\mathbf{x}}_g)\hat{\mathbf{x}}_g$. For the detail of derivation, see Section 2.2.

4. Computational Cost

Our method does not alter the forward pass and thus incurs no cost at test time. For backward pass at training, we observe that, before and after inserting RMPG layers, the backward time for quaternion / 6D / 9D / 10D representations, averaged among 1K iterations on a GeForce RTX 3090, changes from 4.39 / 4.48 / 4.48 / 4.53 to 4.45 / 4.43 / 4.49 / 4.63 (unit: 10^{-2} s), and the memory cost changes from 11449 / 11415 / 10781 / 11363 to 11457 / 11459 / 11545 / 11447 (unit: MiB). Note that the runtime is almost keep the same, as Riemannian optimization only performs an additional projection and we derive and always use 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-

Methods	King's College		Old Hospital		Shop Facade		St Mary's Church		Average	
	T(m)	R($^\circ$)	T(m)	R($^\circ$)	T(m)	R($^\circ$)	T(m)	R($^\circ$)	T(m)	R($^\circ$)
Euler	1.16	2.85	2.54	2.95	1.25	6.48	1.98	6.97	1.73	4.81
Axis-Angle	1.12	2.63	2.41	3.38	**0.84**	5.05	2.16	7.58	1.63	4.66
Quaternion	**0.98**	2.50	2.39	3.44	1.06	6.01	2.59	8.81	1.76	5.19
6D	1.10	2.56	2.21	3.43	1.01	5.43	**1.73**	5.82	1.51	4.31
9D	1.14	3.03	2.11	3.50	0.88	6.39	1.95	5.95	1.52	4.72
9D-Inf	**0.98**	2.32	**1.89**	3.32	1.15	6.36	1.96	6.25	**1.50**	4.56
10D	1.54	2.62	2.32	3.39	1.20	5.76	1.85	6.69	1.73	4.62
RPMG-Quat	1.04	1.91	2.42	2.72	0.98	4.28	1.82	4.89	1.57	3.45
RPMG-6D	1.55	**1.70**	2.62	3.09	0.95	5.01	2.44	5.18	1.89	3.75
RPMG-9D	1.57	1.82	4.37	3.12	0.93	4.17	1.92	**4.69**	2.20	3.45
RPMG-10D	1.30	1.74	3.21	**2.59**	1.10	**3.47**	2.20	5.09	1.95	**3.22**

Table 3. **Camera relocalization on Cambridge Landscape dataset.** We report the *median* error of translation and rotation of the best checkpoint, which is chosen by minimizing the median of rotation. We only care about the rotation error here.

inf is just a special case of our methods with $\lambda = 1$ and $\tau = \tau_{gt}$). But compared to vanilla 4D/6D/9D/10D representation, our methods can still bring a great improvement.

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 $\|\mathbf{R}X - \mathbf{R}_{gt}X\|_F^2$ here, where X is the complete point clouds.

Since the format of loss is changed, the previous schedule of τ is not suitable anymore, and we have to change the value of τ accordingly. Our selection skill is to first choose a τ as we like and visualize the mean geodesic distance between predicted $\mathbf{R}$ and $\mathbf{R}_g$ during training. Then we can roughly adjust τ 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**.

Methods	Mean ($^\circ$)	Med ($^\circ$)	5°Acc (%)
Euler	12.14	6.91	33.6
Axis-Angle	35.49	20.80	4.7
Quaternion	11.54	7.67	29.8
6D	14.13	9.41	23.4
9D	11.44	8.01	23.8
9D-Inf	4.07	3.28	76.7
10D	9.28	7.05	32.6
RPMG-Quat	4.86	3.25	75.8
RPMG-6D	**2.71**	**2.04**	**92.1**
RPMG-9D	3.75	2.10	91.1
RPMG-10D	3.30	2.70	86.8

Table 4. **Flow Loss for Rotation Estimation from Point Clouds.** All models are trained for 30K iterations.

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 them 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.

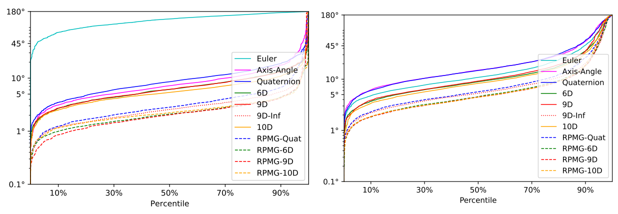


Table 5. **Test Error Percentiles for Experiment 5.1&5.2 in our main paper.** Left: test error percentiles of *airplane* for Experiment 5.1 after training completes. Right: test error percentiles of *chair* for Experiment 5.2 after training completes.

7. Addition on Rotation Representations

Standard mapping between rotation matrix and unit quaternion The *rotation mapping* $\phi : \mathbf{q} \mapsto \mathbf{R}$ algebraically manipulates a unit quaternion $\mathbf{q}$ into a rotation matrix:

$$\phi(\mathbf{q}) = \begin{pmatrix} 2(q_0^2+q_1^2)-1 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & 2(q_0^2+q_2^2)-1 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & 2(q_0^2+q_3^2)-1 \end{pmatrix} \tag{30}$$

where $\mathbf{q} = (q_0, q_1, q_2, q_3) \in \mathcal{S}^3$.

In the reverse direction, the *representation mapping* $\psi(\mathbf{R})$ can be expressed as:

$$\begin{cases} q_0 = \sqrt{1 + R_{00} + R_{11} + R_{22}}/2 \\ q_1 = (R21 - R_{12})/(4 * q_0) \\ q_2 = (R_{02} - R_{20})/(4 * q_0) \\ q_3 = (R_{10} - R_{01})/(4 * q_0) \end{cases} \tag{31}$$

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

References

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009. 1

[2] Earl A Coddington and Norman Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955. 1

[3] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015. 6

[4] Jake Levinson, Carlos Esteves, Kefan Chen, Noah Snavely, Angjoo Kanazawa, Afshin Rostamizadeh, and Ameesh Makadia. An analysis of svd for deep rotation estimation. *arXiv preprint arXiv:2006.14616*, 2020. 5, 7

[5] Valentin Peretroukhin, Matthew Giamou, David M. Rosen, W. Nicholas Greene, Nicholas Roy, and Jonathan Kelly. A Smooth Representation of SO(3) for Deep Rotation Learning with Uncertainty. In *Proceedings of Robotics: Science and Systems (RSS'20)*, Jul. 12–16 2020. 2, 4

[6] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 6

[7] Olinde Rodrigues. Des lois géométriques qui régissent les déplacements d'un système solide dans l'espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire. *Journal de mathématiques pures et appliquées*, 5(1):380–440, 1840. 2

[8] Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015. 5

[9] Camillo J Taylor and David J Kriegman. Minimization on the lie group so (3) and related manifolds. *Yale University*, 16(155):6, 1994. 2

[10] Florian Walch, Caner Hazirbas, Laura Leal-Taixé, Torsten Sattler, Sebastian Hilsenbeck, and Daniel Cremers. Image-based localization using lstms for structured feature correlation. In *ICCV*, October 2017. 6

[11] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition*, 2015. 6

[12] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision*, pages 75–82, 2014. 5

[13] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019. 6