PROJECTIVE MANIFOLD GRADIENT LAYER FOR DEEP ROTATION REGRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Regressing rotations on SO(3) manifold using deep neural networks is an important yet unsolved problem. The gap between Euclidean network output space and the non-Euclidean SO(3) manifold imposes a severe challenge for neural network learning in both forward and backward passes. While several works have proposed different regression-friendly rotation representations, very few works have been devoted to improving the gradient backpropagating in the backward pass. In this paper, we propose a manifold-aware gradient that directly backpropagates into deep network weights. Leveraging the Riemannian gradient and a novel projective gradient, our proposed regularized projective manifold gradient (RPMG) helps networks achieve new state-of-the-art performance in a variety of rotation estimation tasks. Our proposed gradient layer can also be applied to other smooth manifolds such as the unit sphere.

1 INTRODUCTION

Estimating rotations is a crucial problem in visual perception that has broad applications, *e.g.*, in object pose estimation, robot control, camera relocalization, 3D reconstruction and visual odometry (Kendall et al., 2015a; Bui et al., 2020; Wang et al., 2019a; Gojcic et al., 2020; Dong et al., 2020). Recently, with the proliferation of deep neural networks, learning to accurately regress rotations is attracting more and more attention. However, the non-Euclidean characteristics of rotation space make accurately regressing rotation very challenging.

As we know, rotations reside in a non-Euclidean manifold, $\mathrm{SO}(3)$ group, whereas the unconstrained outputs of neural networks usually live in Euclidean spaces. This gap between the neural network output space and $\mathrm{SO}(3)$ manifold becomes a major barrier to accurate rotation regression, thus tackling this gap becomes an important research topic for rotation regression. One popular research direction is to design learning-friendly rotation representations, *e.g.*, 6D continuous representation from (Zhou et al., 2019) and 10D symmetric matrix representation from (Peretroukhin et al., 2020). Recently, (Levinson et al., 2020) adopted the vanilla 9D matrix representation discovering that simply replacing Gram-Schmidt process in the 6D representation (Zhou et al., 2019) by symmetric SVD-based orthogonalization can make this representation superior to the others.

Despite of the progress on discovering better rotation representations, the gap between a Euclidean network out space and $\mathrm{SO}(3)$ manifold hasn't been completely filled. The non-Euclidean nature of $\mathrm{SO}(3)$ manifold leads to many unique properties beyond various representations, one of which is its gradient. For a variable on $\mathrm{SO}(3)$ manifold, its gradient can also be on-manifold, as known as *Riemannian gradient*. We observe that, for gradient backpropagation from the rotation loss back to the neural network weights, all the existing works simply rely upon a vanilla auto-differentiation, yielding off-manifold gradients for predicted rotations. We further point out that most of the existing works focus on a holistic design of rotation regression that is agnostic to forward/backward pass without an in-depth study of its gradient in the backward pass. On one hand, methods of Riemannian optimization allow for optimization on $\mathrm{SO}(3)$ (Taylor & Kriegman, 1994; Blanco, 2010), matrix manifolds (Absil et al., 2009) or general Riemannian manifolds (Zhang et al., 2016; Udriste, 2013). However, they are not very useful when it comes to updating the weights of the neural networks that are Euclidean. On the other hand, approaches like (Hou et al., 2018) incorporate a Riemannian distance as well as its gradient into the network training, however, they do not deal with the *representation* issue.

In this work, we want to *propose a better manifold-aware gradient in the backward pass of rotation regression*. This is a fundamental yet currently under-explored avenue. We begin by making the observation that the gradient of a loss function with respect to the output rotation is often not *on-manifold*. We therefore leverage the Riemannian gradient, connecting the output rotation to a goal rotation in $\mathrm{SO}(3)$. Backpropagating this gradient, we encounter the mapping function (or orthogonalization function) that transforms the raw network output to a valid rotation. This projection is typically a many-to-one map, *e.g.* different matrices can be orthogonalized to the same rotation matrix via either Gram-Schmidt process or SVD orthogonalization. This non-bijectivity provides us with a new design space for our gradient: if we were to use a gradient to update the raw output rotation, many gradients would result in the same update in the final output rotation despite being completely different for backpropagating into the neural network weights. This in fact becomes a supervision problem: *which gradient is the best for backpropagation when many of them correspond to the same update to the output?*

We observe that this problem is somewhat similar to some ambiguities or multi-ground-truth issues. One example would be the symmetry issue in pose estimation: an symmetric object, e.g. a textureless cube, appears the same under many different poses, which needs to be considered when supervising the pose predictions. For supervising such learning problem, one can leverage an uncertainty-driven approach that predicts multimodal distribution rather than a single mode (Deng et al., 2020), however this approach is not feasible for us to get the best gradient. Inspired by the min-of-N loss proposed by (Fan et al., 2017) and used by (Wang et al., 2019b) for dealing with pose symmetry, we propose to find the gradient with the smallest norm that can update the final output rotation to the goal rotation. This *back-projection* process involves finding an element closest to the network output in the inverse image of the goal rotation and projecting the network output to this inverse image space. We therefore coin our gradient *projective manifold gradient*. One thing to note is that this projective gradient tends to shorten the network output, causing the norms of network output vanishing. To fix this problem, we further incorporate a simple regularization into the gradient, leading to our full solution *regularized projective manifold gradient*.

Note that our proposed gradient layer operates on the raw network output and can be directly backpropagated into the network weights. Our method is very general and is not tied to a specific rotation representation. It can be coupled with different non-Euclidean rotation representations, including quaternion, 6D representation (Zhou et al., 2019), and 9D rotation matrix representation (Levinson et al., 2020), and can even be used for regressing other non-manifold variables.

We evaluate our devised projective manifold gradient layers on a diverse set of problems involving rotation regression: 3D object pose estimation from 3D point clouds/images, rotation estimation problems without using ground truth rotation supervisions, and self-supervised instance-level rotation estimation (see appendix D for more experiments on camera relocalization). Our method demonstrates significant and consistent improvements on all these tasks and different all rotation representations tested. Going beyond rotation estimation, in appendix D.3 we also demonstrate performance improvements on regressing unit vectors (lie on a unit sphere) as an example of an extension to other non-Euclidean manifolds.

2 PRELIMINARIES

2.1 RIEMANNIAN GEOMETRY

We define an m-dimensional *Riemannian manifold* embedded in an ambient Euclidean space $\mathcal{X} = \mathbb{R}^d$ and endowed with a *Riemannian metric* $\mathbf{G} \triangleq (\mathbf{G}_\mathbf{x})_{\mathbf{x}\in\mathcal{M}}$ to be a smooth curved space $(\mathcal{M}, G)$. A vector $\mathbf{v} \in \mathcal{X}$ is said to be *tangent* to $\mathcal{M}$ at $\mathbf{x}$ iff there exists a smooth curve $\gamma : [0, 1] \mapsto \mathcal{M}$ s.t. $\gamma(0) = \mathbf{x}$ and $\dot{\gamma}(0) = \mathbf{v}$. The velocities of all such curves through $\mathbf{x}$ form the *tangent space* $\mathcal{T}_\mathbf{x}\mathcal{M} = \{\dot{\gamma}(0) \,|\, \gamma : \mathbb{R} \mapsto \mathcal{M} \text{ is smooth around } 0 \text{ and } \gamma(0) = \mathbf{x}\}$. The Riemannian metric $G(\cdot)$ equips each point $\mathbf{x}$ with an inner product in the tangent space $\mathcal{T}_\mathbf{x}\mathcal{M}$, *e.g.* $\langle \mathbf{u}, \mathbf{v}\rangle_x = \mathbf{u}^T\mathbf{G}_\mathbf{x}\mathbf{v}$.

Definition 1 (Riemannian gradient)**.** *For a smooth function $f : \mathcal{M} \mapsto \mathbb{R}$ and $\forall(\mathbf{x}, \mathbf{v}) \in \mathcal{T}\mathcal{M}$, we define the* Riemannian gradient *of f as the unique vector field $\mathrm{grad} f$ satisfying (Boumal, 2020):*

$$\mathrm{D}f(x)[\mathbf{v}] = \langle \mathbf{v}, \mathrm{grad} f(\mathbf{x})\rangle_x \tag{1}$$

where $\mathrm{D}f(x)[\mathbf{v}]$ is the derivation of f by $\mathbf{v}$. It can further be shown (see our appendix) that an expression for $\mathrm{grad} f$ can be obtained through the projection of the classical *gradient orthogonally*

onto the tangent space

$$\operatorname{grad} f(\mathbf{x}) = \nabla f(\mathbf{x})_{||} = \Pi_{\mathbf{x}}\big(\nabla f(\mathbf{x})\big). \tag{2}$$

where $\Pi_{\mathbf{x}} : \mathcal{X} \mapsto \mathcal{T}_{\mathbf{x}}\mathcal{M} \subseteq \mathcal{X}$ *is an orthogonal projector with respect to* $\langle \cdot, \cdot \rangle_{\mathbf{x}}$.

Definition 2 (Riemannian optimization)**.** *We consider first order optimizers to solve problems of the form* $\min_{\mathbf{x}\in\mathcal{M}} f(\mathbf{x})$*. For a local minimizer or a* stationary point $\mathbf{x}^\star$ *of* f*, the Riemannian gradient vanishes* $\operatorname{grad} f(\mathbf{x}^\star) = 0$ *enabling a simple algorithm,* Riemannian gradient descent *(RGD):*

$$\mathbf{x}_{k+1} = R_{\mathbf{x}_k}(-\tau_k \operatorname{grad} f(\mathbf{x}_k)) \tag{3}$$

where τ_k *is the step size at iteration* k *and* $R_{\mathbf{x}_k}$ *is the* retraction *usually chosen related to the exponential map.*

2.2 Rotation representations

There are many ways of representing a rotation: classic rotation representations, *e.g.* Euler angles, axis-angle, and quaternion; and recently introduced regression-friendly rotation representations such as *e.g.* 5D (Zhou et al., 2019), 6D representations (Zhou et al., 2019) and 10D (Peretroukhin et al., 2020) representations. A majority of deep neural networks can output an *unconstrained*, arbitrary n-dimensional vector $\mathbf{x}$ in a Euclidean space $\mathcal{X} = \mathbb{R}^n$. For Euler angle and axis-angle representations which use a vector from $\mathbb{R}^3$ to represent a rotation, a neural network can simply output a 3D vector; however, for quaternions or 6D and 9D representations that lie on non-Euclidean manifolds, manifold mapping functions $\pi : \mathbb{R}^n \mapsto \mathcal{M}$ are generally needed for normalization or orthogonalization purposes to convert network outputs to valid elements belong to the representation manifold. This network Euclidean output space $\mathcal{X}$ is where the representation manifolds reside and therefore are also called ambient space.

Definition 3 (Rotation representation)**.** *One rotation representation, which lies on a representation manifold* $\mathcal{M}$*, defines a surjective rotation mapping* $\phi : \hat{\mathbf{x}} \in \mathcal{M} \rightarrow \phi(\hat{\mathbf{x}}) \in \mathrm{SO}(3)$ *and a representation mapping function* $\psi : \mathbf{R} \in \mathrm{SO}(3) \rightarrow \psi(\mathbf{R}) \in \mathcal{M}$*, such that* $\phi(\psi) = \mathbf{R} \in \mathrm{SO}(3)$.

Definition 4 (Manifold mapping function)**.** *From an ambient space* $\mathcal{X}$ *to the representation manifold* $\mathcal{M}$*, we can define a manifold mapping function* $\pi : \mathbf{x} \in \mathcal{X} \rightarrow \pi(\mathbf{x}) \in \mathcal{M}$*, which projects a point* $\mathbf{x}$ *in the ambient, Euclidean space to a valid element* $\hat{\mathbf{x}} = \pi(\mathbf{x})$ *on the manifold* $\mathcal{M}$.

We summarize the rotation mappings, representation mappings, the manifold mappings for several non-Euclidean rotation representations below.

Unit quaternion. Unit quaternions represent a rotation using a 4D unit vector $\mathbf{q} \in \mathcal{S}^3$ double covering the non-Euclidean 3-sphere *i.e.* $\mathbf{q}$ and $-\mathbf{q}$ identify the same rotation. A network with a final linear activation can only predict $\mathbf{x} \in \mathbb{R}^4$. The corresponding manifold mapping function is usually chosen to be a normalization step, which reads $\pi_q(\mathbf{x}) = \mathbf{x}/|\mathbf{x}|$. For rotation and representation mapping, we leverage the standard mappings between rotation and quaternion (see appendix F).

6D rotation representation and Gram-Schmidt orthogonalization. 6D rotation representation, proposed in (Zhou et al., 2019), uses two orthogonal unit 3D vectors $(\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2)$ to represent a rotation, which are essentially the first two columns of the corresponding rotation matrix. This representation lies on Stiefel manifold $\mathcal{V}_2(\mathbb{R}^3)$. Its manifold mapping π_{6D} is done through Gram-Schdimt orthogonalization. Its rotation mapping ϕ_{6D} is done by adding the third column $\hat{\mathbf{c}}_3 = \hat{\mathbf{c}}_1 \times \hat{\mathbf{c}}_2$. Its representation mapping ψ_{6D} is simply getting rid of the third column $\hat{\mathbf{c}}_3$ from a rotation matrix.

9D rotation matrix representation and SVD orthogonalization. As this representation manifold is $\mathrm{SO}(3)$, both the rotation and representation mapping functions are simply identity. To map a raw 9D network output $\mathbf{M}$ to a rotation matrix, we can consider both Gram-Schmidt orthogonalization and SVD orthogonalization. However, for Gram-Schmidt orthogonalization, the last three dimensions are redundant and will not make any difference with 6D representation. We therefore choose SVD orthogonalization as the manifold mapping function π_{9D}, as follows: π_{9D} first decomposes $\mathbf{M}$ into its left and right singular vectors $(\mathbf{U}, \mathbf{V}^\top)$ and singular values (SV) Σ, $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$; then it replaces the SV $\Sigma \leftarrow \Sigma' = diag(1, 1, det(\mathbf{U}\mathbf{V}^\top))$ and finally, computes $\mathbf{R} = \mathbf{U}\Sigma'\mathbf{V}^\top$ to get the corresponding rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$.

2.3 Deep rotation regression

We conclude this section by describing the ordinary forward and backward passes of a neural network based rotation regression, as used in (Zhou et al., 2019; Levinson et al., 2020).

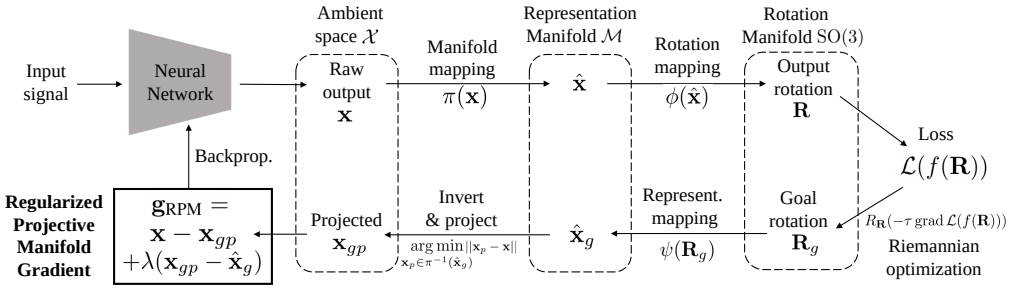

Figure 1: **Projective Manifold Gradient Layer.** In the forward pass, the network predicts a raw output $\mathbf{x}$, which is then transformed into a valid rotation $\mathbf{R} = \phi(\pi(\mathbf{x}))$. We leave this forward pass unchanged and only modify the backward pass. In the backward pass, we first use Riemannian optimization to get a goal rotation $\mathbf{R}_g$ and map it back to $\hat{\mathbf{x}_g}$ on the representation manifold $\mathcal{M}$. After that we find the element $\mathbf{x}_{gp}$ which is closest to the raw output in the inverse image of $\hat{\mathbf{x}_g}$, and finally get the gradient $\mathbf{g}_{\mathbf{RPM}}$ we want.

Forward and backward passes. Assume, for a rotation representation, the network predicts $\mathbf{x} \in \mathcal{X}$, then the manifold mapping π will map $\mathbf{x}$ to $\hat{\mathbf{x}} = \pi(\mathbf{x}) \in \mathcal{M}$, followed by a rotation mapping ϕ that finally yields the output rotation $\mathbf{R} = \phi(\hat{\mathbf{x}}) = \phi(\pi(\mathbf{x}))$. Our work only tackles the backward pass and keeps the forward pass unchanged, as shown in the top part of Figure 1. The gradient in the backward-pass is simply computed using Pytorch autograd method, that is $\mathbf{g} = f'(\mathbf{R})\phi'(\hat{\mathbf{x}})\pi'(\mathbf{x})$.

Loss function. For supervising rotation matrix, the most common choice of loss function is L2 loss, $||\mathbf{R} - \mathbf{R}_{gt}||_F^2$, as used by (Zhou et al., 2019; Levinson et al., 2020). This loss is equal to $4 - 4\cos(< \mathbf{R}, \mathbf{R}_{gt} >)$, where $< \mathbf{R}, \mathbf{R}_{gt} >$ represents the angle between $\mathbf{R}$ and $\mathbf{R}_{gt}$.

3 METHOD

Overview. In this work, we propose a *projective manifold gradient layer*, without changing the forward pass of a given rotation regressing network, as shown in Figure 1. Our focus is to find a better gradient $\mathbf{g}$ of the loss function $\mathcal{L}$ with respect to the network raw output $\mathbf{x}$ for backpropagation into the network weights.

Let's start with examining the gradient of network output x in a general case – regression in Euclidean space. Given a ground truth $\mathbf{x}_{gt}$ and the L2 loss $||\mathbf{x} - \mathbf{x}_{gt}||^2$ that maximizes the likelihood in the presence of Gaussian noise in $\mathbf{x}$, the gradient would be $\mathbf{g} = 2(\mathbf{x} - \mathbf{x}_{gt})$.

In the case of rotation regression, we therefore propose to find a proper $\mathbf{x}^* \in \mathcal{X}$ for a given ground truth $\mathbf{R}_{gt}$ or a computed goal rotation $\mathbf{R}_g$ when the ground truth rotation is not available, and then simply use $\mathbf{x} - \mathbf{x}^*$ as our gradient to backpropagate into the network.

Note that finding such a $\mathbf{x}^*$ can be challenging. Assuming we know $\mathbf{R}_{gt}$, finding a $\mathbf{x}^*$ involves inverting ϕ and π since the network output $\mathbf{R} = \phi(\pi(\mathbf{x}))$. Furthermore, we may not know $\mathbf{R}_{gt}$ under indirect rotation supervision (*e.g.*, flow loss as used in PoseCNN(Xiang et al., 2017)) and self-supervised rotation estimation cases (*e.g.*, 2D mask loss as used in (Wang et al., 2020)). In this work, we introduce the following techniques to mitigate these problems: (i) we first take a Riemannian gradient to compute a goal rotation $\mathbf{R}_g \in \mathrm{SO}(3)$, which does not rely on knowing $\mathbf{R}_{gt}$, as explained in Section 3.1; (ii) we then find the set of all possible $\mathbf{x}_g$s that can be mapped to $\mathbf{R}_g$, or in other words, the inverse image of $\mathbf{R}_g$ under π and ϕ; (iii) we find $\mathbf{x}_{gp}$ which is the element in this set closet to $\mathbf{x}$ in the Euclidean metric and set it as "$\mathbf{x}^*$". We will construct our gradient using this $\mathbf{x}^*$, as explained in 3.2. (iv) we add a regularization term to this gradient forming $\mathbf{g}_{RPMG}$ as explained in 3.3. The whole backward pass leveraging our proposed regularized projective manifold gradient is shown in the lower half of Figure 1.

3.1 RIEMANNIAN GRADIENT AND GOAL ROTATION $\mathbf{R}_g$

To handle rotation estimation with/without direct rotation supervision, we first propose to compute the Riemannian gradient of the loss function $\mathcal{L}$ with respect to the output rotation $\mathbf{R}$ and find a goal rotation $\mathbf{R}_g$ that is presumably closer to the ground truth rotation than $\mathbf{R}$.

Assume the loss function is in the following form $\mathcal{L}(f(\mathbf{R}))$, where $\mathbf{R} = \pi(\phi(\mathbf{x}))$ is the output rotation and f constructs a loss function that compares $\mathbf{R}$ to the ground truth rotation $\mathbf{R}_{gt}$ directly or indirectly. Given $\mathbf{R}(\mathbf{x})$ and $\mathcal{L}(f(\mathbf{R}(\mathbf{x})))$, we can perform one step of Riemannian optimization yielding our goal rotation $\mathbf{R}_g \leftarrow R_{\mathbf{R}}(-\tau \operatorname{grad} \mathcal{L}(f(\mathbf{R})))$, where τ is the step size of Riemannian gradient and can be set to a constant as a hyperparameter or varying during the training. For L2 loss $||\mathbf{R} - \mathbf{R}_{gt}||_F^2$, Riemannian gradient is always along the geodesic path between $\mathbf{R}$ and $\mathbf{R}_{gt}$ on SO(3). In this case, $\mathbf{R}_g$ can generally be seen as an intermediate goal between $\mathbf{R}$ and $\mathbf{R}_{gt}$ dependent on τ. Gradually increasing τ from 0 will first make $\mathbf{R}_g$ approach $\mathbf{R}_{gt}$ starting with $\mathbf{R}_g = \mathbf{R}$, and then reach $\mathbf{R}_{gt}$ where we denote $\tau = \tau_{gt}$, and finally going beyond $\mathbf{R}_{gt}$. Although, when $\mathbf{R}_{gt}$ is available, one can simply set $\mathbf{R}_g = \mathbf{R}_{gt}$, we argue that this is just a special case under $\tau = \tau_{gt}$. We will further compare different strategies of how to choose or vary τ in Section 3.3. For scenarios where $\mathbf{R}_{gt}$ is unavailable, $\mathbf{R}_g$ is presumably closer to the desired $\mathbf{R}_{gt}$ if the loss function does not suffer from local minima. In the sequel, we only use $\mathbf{R}_g$ for explaining our methods.

3.2 Projective Manifold Gradient

Given $\mathbf{R}_g$, we can use the representation mapping ψ to find the corresponding $\hat{\mathbf{x}}_g = \psi(\mathbf{R}_g)$ on the representation manifold $\mathcal{M}$. However, further inverting π and finding the corresponding $\mathbf{x}_g \in \mathcal{X}$ is a non-trivial problem, due to the projective nature of π. In fact, there are many $\mathbf{x}_g$s that satisfy $\pi(\mathbf{x}_g) = \hat{\mathbf{x}}_g$. It seems that we can construct a gradient $\mathbf{g} = (\mathbf{x} - \mathbf{x}_g)$ using any $\mathbf{x}_g$ that satisfies $\pi(\mathbf{x}_g) = \hat{\mathbf{x}}_g$. No matter which $\mathbf{x}_g$ we choose, if this gradient were to update $\mathbf{x}$, it will result in the same $\mathbf{R}_g$. But, when backpropagation into the network, those gradients will update the network weights differently, potentially resulting in different learning efficiency and network performance.

Formally, we formulate this problem as *a multi-ground-truth problem* for x: we need to find the best x^* to supervise from the inverse image of $\hat{\mathbf{x}}_g$ under the mapping π. We note that similar problems have been seen in pose supervision dealing with symmetry as in (Wang et al., 2019b), where one needs to find one pose to supervise when there are many poses under which the object appears the same. (Wang et al., 2019b) proposed to use a min-of-N strategy introduced by (Fan et al., 2017): from all possible poses, taking the pose that is closet to the network prediction as ground truth. A similar strategy is also seen in supervising quaternion regression, as q and $-q$ stand for the same rotation. One common choice of the loss function is therefore $\min\{\mathcal{L}(q, q_{gt}), \mathcal{L}(q, -q_{gt})\}$(Peretroukhin et al., 2020), which penalizes the distance to the closest ground truth quaternion.

Inspired by these works, we propose to choose our gradient among all the possible gradients with the lowest level of redundancy , *i.e.*, we require x^* to be the one closet to x, or in other words, the gradient to have the smallest norm, meaning that we need to find the projection point $\mathbf{x}_{gp}$ of $\mathbf{x}$ to all the valid $\mathbf{x}_g$:

$$\mathbf{x}_{gp} = \underset{\pi(\mathbf{x}_g)=\hat{\mathbf{x}}_g}{\operatorname{argmin}} \; ||\mathbf{x} - \mathbf{x}_g||_2 \tag{4}$$

We then can construct our *projective manifold gradient* as $\mathbf{g}_{PM} = \mathbf{x} - \mathbf{x}_{gp}$.

Here we provide another perspective why a network may prefer such a gradient. In the case where a deep network is trained using stochastic gradient descents (SGD), the final gradient used to update the network weights is averaged across the gradients of all the batch instances. If gradients from different batch instances contain different levels of redundancy, then the averaged gradient may be biased or not even appropriate. This argument is generally applicable to all stochastic optimizers (*e.g.*, Adam (Adams et al., 2020))

Inverting π. There are many ways to solve this projection problem for different manifold mapping functions π. For example, we can formulate this as a constrained optimization problem. For the manifold mapping functions we consider, we propose the following approach: we first solve for the inverse image $\pi^{-1}(\hat{\mathbf{x}}_g)$ of $\hat{\mathbf{x}}_g$ in the ambient space $\mathcal{X}$ analytically, which reads $\pi^{-1}(\hat{\mathbf{x}}_g) = \{\mathbf{x}_g \in \mathcal{X} \mid \pi(\mathbf{x}_g) = \hat{\mathbf{x}}_g\}$; we then project $\mathbf{x}$ onto this inverse image space. Note that, sometimes only a superset of this inverse image can be found analytically, requiring certain constraints on $\mathbf{x}_{gp}$ to be enforced.

Here we list the inverse image of $\pi^{-1}(\hat{\mathbf{x}}_g)$ and the projection point $\hat{\mathbf{x}}_g$ for different rotation representations and their corresponding manifold mapping functions π. Please refer to appendix for the detailed derivations.

Quaternion. With $\pi_q(\mathbf{x}) = \mathbf{x}/||\mathbf{x}||$, $\mathbf{x} \in \mathbb{R}^4$, and $\hat{\mathbf{x}}_g \in \mathcal{S}^3$: $\pi_q^{-1}(\hat{\mathbf{x}}_g) = \{\mathbf{x} \mid \mathbf{x} = k\hat{\mathbf{x}}_g, k \in$

$\mathbb{R}$ and $k > 0\}$, which is a ray in the direction of $\hat{\mathbf{x}}_g$ starting from the origin. Without considering the constraint of $k > 0$, an analytical solution to this projection point $\mathbf{x}_{gp}$ of $\mathbf{x}$ onto this line can be derived: $\mathbf{x}_{gp} = (\mathbf{x} \cdot \hat{\mathbf{x}}_g)\hat{\mathbf{x}}_g$.

6D representation. With π_{6D} as Gram-Schmidt process, $\mathbf{x} = [\mathbf{u}, \mathbf{v}] \in \mathbb{R}^6$, and $\hat{\mathbf{x_g}} \in \mathcal{V}_2(\mathbb{R}^3)$: $\pi_{6D}^{-1}(\hat{\mathbf{x}}_g) = \{[k_1\hat{\mathbf{u}}_g, k_2\hat{\mathbf{u}}_g + k_3\hat{\mathbf{v}}_g] \mid k_1, k_2, k_3 \in \mathbb{R} \text{ and } k_1, k_3 > 0\}$ (the former is a ray whereas the latter spans a half plane). Without considering the constraint of $k_1, k_3 > 0$, the projection point $\mathbf{x}_{gp}$ can be analytically represented as $\mathbf{x}_{gp} = [(\mathbf{u} \cdot \hat{\mathbf{u}}_g)\hat{\mathbf{u}}_g, (\mathbf{v} \cdot \hat{\mathbf{u}}_g)\hat{\mathbf{u}}_g + (\mathbf{v} \cdot \hat{\mathbf{v}}_g)\hat{\mathbf{v}}_g]$

9D representation. With $\pi_{9D}(\mathbf{x})$ as SVD orthogonalization to be positive, $\mathbf{x} \in \mathbb{R}^{3\times3}$, and $\hat{\mathbf{x_g}} \in$ SO(3), the analytical expression for π_{9D}^{-1} is available when we ignore the positive singular value constraints, which gives $\pi_{9D}^{-1}(\hat{\mathbf{x}}_g) = \{\mathbf{S}\hat{\mathbf{x}}_g \mid \mathbf{S} \text{ is an arbitrary symmetric matrix}\}$. We can further solve the projection point $\mathbf{x}_{gp}$ with an elegant representation $\mathbf{x}_{gp} = \frac{\mathbf{x}\hat{\mathbf{x}}_g^T + \hat{\mathbf{x}}_g\mathbf{x}^T}{2}$.

3.3 Regularized projective manifold gradient for rotation regression

Issues in naive projective manifold gradient. In Figure 2, we illustrate this projection process for several occasions where $\mathbf{x}$ takes different positions relative to $\mathbf{x}_g$. We demonstrate that there are two issues in this process. First, no matter where $\mathbf{x}$ is in, the projection operation will shorten the length of our prediction because $|\mathbf{x}_{gp}| < |\mathbf{x}|$ is always true. This will cause the length norm of the prediction of the network to become very small as the training progresses (see Figure 3). The shrinking network output will keep increasing the effective learning rate, preventing the network from convergence and leading to great harm to the network performance (see Table 3 and Figure 3 for ablation study).

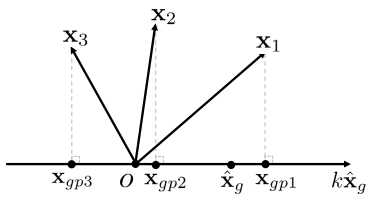

Figure 2: Projection point $\hat{\mathbf{x}}_{gp}$ in the case of quaternion.

Second, when the angle between $\mathbf{x}$ and $\hat{\mathbf{x}}_g$ becomes larger than $\pi/2$ (in the case of $\mathbf{x} = \mathbf{x}_3$), the naive projection $\mathbf{x}_{gp}$ will be in the opposite direction of $\hat{\mathbf{x}}_g$ and can not be mapped back to $\hat{\mathbf{x}}_g$ under π_q, resulting in a wrong gradient. The same set of issues also happen to 6D and 9D representations. The formal reason is that the analytical solution of the inverse image assumes certain constraints are satisfied, which is usually true only when either $\hat{\mathbf{x}}_g$ is not far from $\mathbf{x}$ or the network is about to converge.

Regularized projective manifold gradient To solve the first issue, we propose to add a regularization term $\mathbf{x}_{gp} - \hat{\mathbf{x}}_g$ to the projective manifold gradient, which can avoid the length vanishing problem. The *regularized projective manifold gradient* then reads:

$$\mathbf{g}_{RPM} = \mathbf{x} - \mathbf{x}_{gp} + \lambda(\mathbf{x}_{gp} - \hat{\mathbf{x}}_g), \tag{5}$$

where λ is a regularization coefficient. We intentionally keep the weight of λ, small (usually 0.01) because: (1) we want the projective manifold gradient $(\mathbf{x} - \mathbf{x}_{gp})$ to be the major component of our gradient; (2) since this regularization is roughly proportional to the difference in prediction length and 1, a small lambda can already prevent the length from being vanished and, at the end, the prediction length will stay roughly constant at the equilibrium under projection and regularization.

To tackle the second problem of reversed gradient, we further propose to take a small τ used in Riemannian optimization at the beginning of training, leading to a slow warm-up. As the training progresses, we increase τ, such that the network converges better. Our ablation study will show the effectiveness of our choice of λ and τ.

4 Experiments

We investigate popular rotation representations and find our methods greatly improve the performance in different kinds of tasks. For our regularized projective manifold gradient (in short **RPMG**), we apply it in the backpropagation process of Quaternion, 6D and 9D, without changing the forward pass, leading to three new methods **RPMG-Quat**, **RPMG-6D** and **RPMG-9D**. We compare the following seven baselines: **Euler angle**, **axis-angle**, **Quaternion**, **6D** (Zhou et al., 2019), **9D** (Levinson et al., 2020), **9D-Inf** (Levinson et al., 2020) and **10D** (Peretroukhin et al., 2020). We adopt three evaluation metrics: mean, median, and 5° accuracy of (geodesic) errors between predicted rotation

and ground truth rotation. For most of our experiments, we set the regularization term $\lambda = 0.01$ and increase τ from 0.05 to 0.25 by uniform steps. We further show and discuss the influence of different choices of these two hyperparameters in the ablation studies.

4.1 3D OBJECT POSE ESTIMATION FROM POINT CLOUDS

The first experiment is the category-level pose estimation from point clouds. Given one shape point clouds of a specific category, the network learns to predict the 3D rotation of the input point clouds from the predefined canonical view of this category(Wang et al., 2019b). We replace the point clouds alignment task used in (Zhou et al., 2019; Levinson et al., 2020) (which has almost been solved) by this experiment since it is more challenging and more closed to real-world applications (no canonical point clouds is given to the network).

We use a PointNet++ (Qi et al., 2017) network as our backbone, supervised by L2 loss between the predicted rotation matrix $\mathbf{R}$ and the ground truth rotation matrix $\mathbf{R}_{gt}$. We use two different kinds of data: *complete* point clouds and *depth* point clouds, both generated from the airplane point clouds from ModelNet (Wu et al., 2015). We divide the airplanes into a train split and a test split, following (Chen et al., 2021). Refer to appendix for more details.

The results are shown in Table 1 and Table 2. We see a great improvement of our methods in all three rotation representations. One may find **9D-Inf** also leads to a good performance, which is actually a special case of our method with $\tau = \tau_{gt}$ and $\lambda = 1$. However, this simple loss may lead to a bad performance when R_{gt} is unavailable(see Sec 4.3.2).

	Mean (°)	Med (°)	5°Acc (%)
Euler	124.88	130.37	0.0
Axis-Angle	10.57	7.20	29.1
Quaternion	10.55	8.57	21.7
6D	7.25	6.15	38.2
9D	7.70	6.16	38.0
9D-Inf	3.25	2.57	87.1
10D	7.42	5.34	44.5
RPMG-Quat	5.12	2.83	80.4
RPMG-6D	**2.85**	2.09	**92.7**
RPMG-9D	3.97	**2.02**	92.5

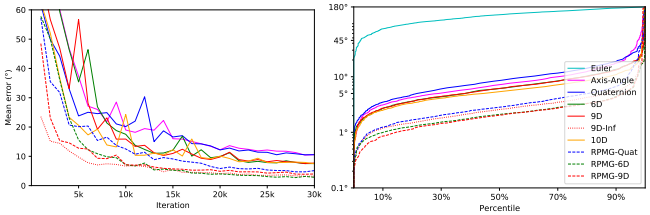

Table 1: **Pose estimation from *complete* point clouds.** Left: a comparison of methods by mean, median, and 5° accuracy of (geodesic) errors after 30k training steps. Middle: mean test error at different points along with the training progression. Right: test error percentiles after training completes. The legend on the right applies to both plots.

	Mean (°)	Med (°)	5°Acc (%)
Euler	126.2	132.7	0.0
Axis-Angle	125.2	132.0	0.0
Quaternion	22.91	15.68	4.8
6D	38.60	25.87	0.9
9D	29.25	20.1	2.4
9D-Inf	**6.39**	4.23	60.2
10D	45.6	31.2	0.9
RPMG-Quat	10.07	5.77	39.9
RPMG-6D	7.10	4.38	57.8
RPMG-9D	6.53	**4.19**	**61.0**

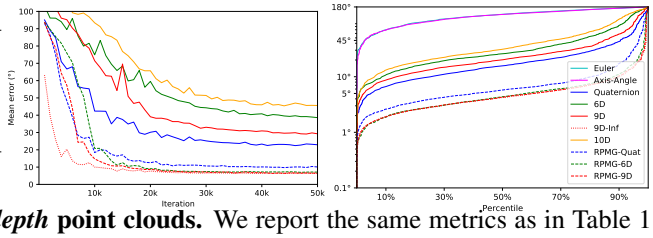

Table 2: **Pose estimation from *depth* point clouds.** We report the same metrics as in Table 1; see the caption there. All models are trained for 50K iterations.

Ablation Study We change different regularization coefficients λ and different τ to show the influence of these two hyperparameters on 6D representation. Our simplest method **MG**, which directly uses $\mathbf{x} - \hat{\mathbf{x}}$ as the gradient, can outperform the vanilla L2 loss but is worse than those with small λ, which is consistent with the discussion in Sec3.3. And we can find that $\lambda = 0.01$ is better than $\lambda = 0$ from the failures of **PMG** which only uses $\mathbf{x} - \mathbf{x}_{gp}$ as gradient. We show the length vanishing problem without regularization and stablized length with regularization in Figure 3. As for the precise value of λ, our experiments show that different choices will lead to similar performances, which implies the robustness of this hyperparameter. For the choices of τ, we also get a similar conclusion as λ: it is not sensitive in a reasonable range, but it shouldn't be too large or too small, just as discussed in Sec 3.3.

4.2 3D OBJECT POSE ESTIMATION FROM REAL IMAGES

Pascal3D+ (Xiang et al., 2014) is a standard benchmark for object pose estimation from real images. We follow the same setting as in (Levinson et al., 2020) to estimate object poses from single images. For training we discard occluded or truncated objects and augment with rendered images from (Su

			Complete			Depth		
			Mean (°)	Med (°)	5°Acc (%)	Mean (°)	Med (°)	5°Acc (%)
L2 w/ 6D	-	-	7.25	6.15	38.2	38.60	25.87	0.9
MG-6D	$\lambda = 1$	τ_{safe}	3.27	2.68	86.1	7.60	4.80	52.6
		τ_{gt}	3.37	2.77	85.7	7.69	4.97	50.6
PMG-6D	$\lambda = 0$	τ_{safe}	64.41	40.99	2.8	92.9	91.5	0.2
		τ_{gt}	103.2	100.4	0.0	132.7	126.5	0.0
RPMG-6D	$\lambda = 0.01$	τ_{init}	3.60	2.30	91.1	**7.02**	4.38	**59.7**
		τ_{safe}	3.41	**2.04**	87.2	23.20	8.71	24.6
		τ_{gt}	4.12	2.22	87.1	23.69	7.77	28.5
		$\tau_{init} \rightarrow \tau_{safe}$	**2.85**	2.09	**92.7**	7.10	4.38	57.8
	$\lambda = 0.005$	$\tau_{init} \rightarrow \tau_{safe}$	3.07	2.11	89.6	7.30	**4.26**	55.0
	$\lambda = 0.1$		**2.85**	2.19	90.9	7.93	4.67	54.7

Table 3: **Ablation study of pose estimation from point clouds.** We report the same metrics as in Table 1; see the caption there. We set $\tau_{init} = 0.05$, $\tau_{safe} = 0.25$. τ_{safe} means the upper bound of τ to make sure the $\mathbf{R}_g$ will be closer to $\mathbf{R}$ than $\mathbf{R}_{gt}$. Refer to appendix for more details.

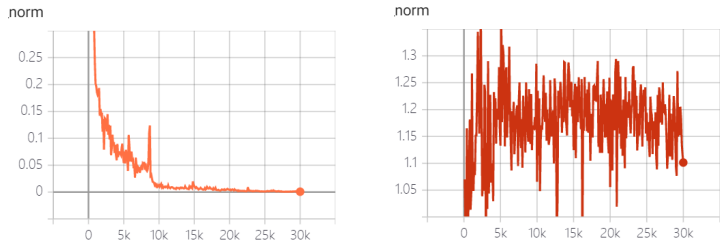

Figure 3: **Average L2 norm of the network raw output** $\mathbf{x}$ during training. Left: PMG-6D (w/o reg. $\lambda = 0$). Right: RPMG-6D (w/ reg. $\lambda = 0.01$)

et al., 2015). In the Table 1 and Table 2, we report our results on *sofa* and *bicycle* categories, given that (Levinson et al., 2020) only reported the detailed numbers for these two categories.

It can be seen that our method leads to consistent improvements to quaternion, 6D, and 9D representations on both sofa and bicycle classes.

	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**

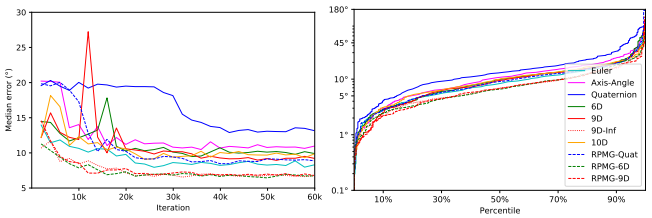

Table 4: **Pose estimation from PASCAL3D+ *sofa* images.** Left: a comparison of methods by 10° / 15° / 20° accuracy of (geodesic) errors and median errors after 60k training steps. Middle: median test error at different points along with the training progression. Right: test error percentiles after training completes. The legend on the right applies to both plots.

4.3 Rotation Estimation without Ground Truth Rotation Supervision

4.3.1 Using Flow Loss for Rotation Estimation from Point Clouds

We mainly follow the setting of experiment 4.1 with complete airplane point cloud 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 fix $\tau = 25$ and $\lambda = 0.01$. In Table 6, we show our methods again outperform vanilla methods.

	Accuracy(%) 10°	15°	20°	Med° 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**

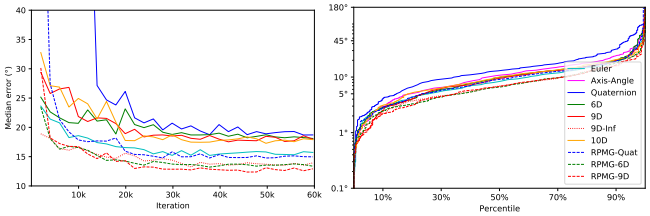

Table 5: **Pose estimation from PASCAL3D+ *bicycle* images.** We report the same metrics as Table 4; see the caption there.

4.3.2 Self-supervised Instance-Level Rotation Estimation from Point Clouds

For one complete chair instance Z, given a complete observation X, we estimate its pose $\mathbf{R}$. We then use chamfer distance between Z and $\mathbf{R}^{-1}X$ as a self-supervised loss. The network structure and training settings are all the same as Experiment 4.1. We simply set $\tau = 1$. The interesting part here is that vanilla **9D-Inf** fails while our methods still perform very well.

	Category-Level Self-Supervise Mean (°)	Med (°)	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

	Instance-Level Self-Supervise Mean (°)	Med (°)	5°Acc (%)
Euler	129.3	132.9	0
Axis-Angle	36.31	6.98	37
Quaternion	4.04	3.30	74
6D	43.9	6.49	44
9D	2.47	2.02	92.5
9D-Inf	101.5	96.61	0
10D	2.18	1.91	96.5
RPMG-Quat	2.88	2.38	91.5
RPMG-6D	3.08	2.92	89.5
RPMG-9D	**1.40**	**1.17**	**100**

Table 6: **Rotation estimation without ground truth rotation supervision** We report the same metrics as in Table 1; see the caption there. All models are trained for 30K iterations. Left: Flow loss for rotation estimation. Right: Self-supervised instance-level rotation estimation.

5 Related Work

Both rotation parameterization and optimization on SO(3) are well-studied topics. Early deep learning models leverages various rotation representations for pose estimation, *e.g.*, axis-angle (Ummenhofer et al., 2017; Do et al., 2018; Gao et al., 2018), quaternion (Xiang et al., 2017; Kendall & Cipolla, 2017; Kendall et al., 2015b) and Euler-angle (Tulsiani & Malik, 2015; Su et al., 2015; Kundu et al., 2018). Recently, (Zhou et al., 2019) points out that Euler-angle, axis-angle, and quaternion are not continuous rotation representations, since their representation spaces are not homeomorphic to SO(3). As better representations for rotation regression, 6D(Zhou et al., 2019), 9D(Levinson et al., 2020), 10D(Peretroukhin et al., 2020) representations are proposed to resolve the discontinuity issue and improve the regression accuracy. A concurrent work (Brégier, 2021) examines different manifold mappings theoretically and experimentally, finding out that SVD orthogonalization performs the best when regressing arbitrary rotations. Originated from general Riemannian optimization, (Taylor & Kriegman, 1994) presents an easy approach for minimization on SO(3) by constructing a local axis-angle parameterization, which is also the tangent space of SO(3) manifold. They backpropagate gradient to the tangent space and use the exponential map to update the current rotation matrix. Most recently, (Teed & Deng, 2021) constructs a PyTorch library that supports tangent space gradient backpropagation for 3D transformation groups, (*e.g.*, SO(3), SE(3), Sim(3)). This proposed library can be used to implement the Riemannian gradient in our layer.

6 Conclusion and Future Work

Our work tackles the problem of designing a gradient layer to facilitate the learning of rotation regression. Our extensive experiments have demonstrated the effectiveness of my method coupled with different rotation representations in diverse tasks dealing with rotation estimation.

REFERENCES

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

Henry Adams, M. Aminian, Elin Farnell, M. Kirby, C. Peterson, Joshua Mirth, R. Neville, P. Shipman, and C. Shonkwiler. A fractal dimension for measures via persistent homology. *arXiv: Dynamical Systems*, pp. 1–31, 2020.

Matthew Grimes Alex Kendall and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. 2015.

Jose-Luis Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3:6, 2010.

Nicolas Boumal. An introduction to optimization on smooth manifolds. *Available online, May*, 2020.

Romain Brégier. Deep regression on manifolds: a 3d rotation case study. *CoRR*, abs/2103.16317, 2021. URL https://arxiv.org/abs/2103.16317.

Mai Bui, Tolga Birdal, Haowen Deng, Shadi Albarqouni, Leonidas Guibas, Slobodan Ilic, and Nassir Navab. 6d camera relocalization in ambiguous scenes via continuous multimodal inference. *arXiv preprint arXiv:2004.04807*, 2020.

Haiwei Chen, Shichen Liu, Weikai Chen, Hao Li, and Randall Hill. Equivariant point network for 3d point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14514–14523, 2021.

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

Haowen Deng, Mai Bui, Nassir Navab, Leonidas Guibas, Slobodan Ilic, and Tolga Birdal. Deep bingham networks: Dealing with uncertainty and ambiguity in pose estimation. *arXiv preprint arXiv:2012.11002*, 2020.

Thanh-Toan Do, Ming Cai, Trung Pham, and Ian D. Reid. Deep-6dpose: Recovering 6d object pose from a single RGB image. *CoRR*, abs/1802.10367, 2018. URL http://arxiv.org/abs/1802.10367.

Siyan Dong, Qingnan Fan, He Wang, Ji Shi, Li Yi, Thomas Funkhouser, Baoquan Chen, and Leonidas Guibas. Robust neural routing through space partitions for camera relocalization in dynamic indoor environments. *arXiv preprint arXiv:2012.04746*, 2020.

Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 605–613, 2017.

Ge Gao, Mikko Lauri, Jianwei Zhang, and Simone Frintrop. Occlusion resistant object rotation regression from point cloud segments. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, September 2018.

Zan Gojcic, Caifa Zhou, Jan D Wegner, Leonidas J Guibas, and Tolga Birdal. Learning multiview 3d point cloud registration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1759–1769, 2020.

Benjamin Hou, Nina Miolane, Bishesh Khanal, Matthew CH Lee, Amir Alansary, Steven McDonagh, Jo V Hajnal, Daniel Rueckert, Ben Glocker, and Bernhard Kainz. Computing cnn loss and gradients for pose estimation with riemannian geometry. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 756–764. Springer, 2018.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

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*, pp. 2938–2946, 2015a.

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 2015b.

Abhijit Kundu, Yin Li, and James M. Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

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.

Shuai Liao, Efstratios Gavves, and Cees G. M. Snoek. Spherical regression: Learning viewpoints, surface normals and 3d rotations on n-spheres. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Long Beach, USA, June 2019.

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.

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.

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.

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.

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

Zachary Teed and Jia Deng. Tangent space backpropagation for 3d transformation groups. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

Constantin Udriste. *Convex functions and optimization methods on Riemannian manifolds*, volume 297. Springer Science & Business Media, 2013.

Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. Demon: Depth and motion network for learning monocular stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

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. URL `https://github.com/NavVisResearch/NavVis-Indoor-Dataset`.

Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3343–3352, 2019a.

Gu Wang, Fabian Manhardt, Jianzhun Shao, Xiangyang Ji, Nassir Navab, and Federico Tombari. Self6d: Self-supervised monocular 6d object pose estimation. In *The European Conference on Computer Vision (ECCV)*, August 2020.

He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2642–2651, 2019b.

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.

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*, pp. 75–82, 2014. doi: 10.1109/WACV.2014.6836101.

Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

Hongyi Zhang, Sashank J Reddi, and Suvrit Sra. Riemannian svrg: Fast stochastic optimization on riemannian manifolds. *arXiv preprint arXiv:1605.07147*, 2016.

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*, pp. 5745–5753, 2019.

A ADDITION ON PRELIMINARIES

A.1 MORE ON RIEMANNIAN GEOMETRY

In this part, we supplement the definitions in Section 2.3 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 (Coddington & Levinson, 1955), given any $(\mathbf{x}, \mathbf{v}) \in \mathcal{TM}$, there exists a unique *maximal*[1] 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: $\mathrm{Exp}_{\mathbf{x}}(\mathbf{v}) = \gamma_{\mathbf{v}}(1)$. We will refer to the well-defined, smooth inverse of this map as the *logaritmic map*: $\mathrm{Log}_{\mathbf{x}}\mathbf{y} \triangleq \mathrm{Exp}_{\mathbf{x}}^{-1}(\mathbf{v})$. 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* $\pi : \mathcal{X} \mapsto \mathcal{M}$ and a *tangent space projector* $\Pi_{\mathbf{x}} : \mathcal{X} \mapsto \mathcal{T}_{\mathbf{x}}\mathcal{M}$ both 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 (Absil et al., 2009).

A.2 10D SYMMETRIC MATRIX REPRESENTATION.

(Peretroukhin et al., 2020) presents an alternative representation of a quaternion: instead of predicting a 4D vector, $\mathbf{q}$ is over-parameterized by a 4×4 symmetric matrix. The eigen-vector of this matrix with the smallest eigenvalue constitutes the map from this representation into a unit quaternion. We find that this variant of quaternions imposes obstacles in finding the inverse image of the manifold mapping.

[1] *maximal* refers to the fact that the curve is as long as possible.

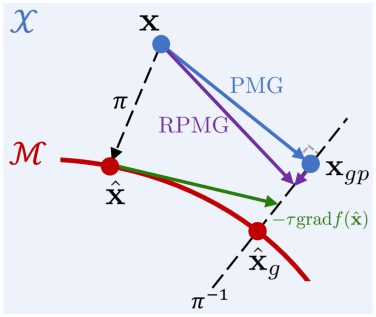

Figure 4: **Illustration for regularized projective manifold gradient.** First we project $\mathbf{x}$ to $\hat{\mathbf{x}}$ by π, and compute a Riemannian gradient, which is shown as the *green* arrow. After getting a next goal $\hat{\mathbf{x}}_g \in \mathcal{M}$ by Riemannian gradient, we find the inverse projection $\mathbf{x}_{gp}$ of $\hat{\mathbf{x}}_g$, which leads to our *projective manifold gradient*, shown as the *blue* arrow. With a regularization term, we can get our final *regularized projective manifold gradient*, as the *purple* arrow.

B Projective Manifold Gradient on SO(3)

B.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 $GL(3)$:

$$\text{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times 3} : \mathbf{R}^\top \mathbf{R} = \mathbf{I}, \det(\mathbf{R}) = 1\}. \tag{6}$$

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 $\mathbf{\Omega} \in \mathcal{T}_\mathbf{I}\mathcal{M}$ can be uniquely mapped to the manifold $\mathcal{M}$ through the exponential map:

$$\text{Exp}_\mathbf{I}(\mathbf{\Omega}) = \mathbf{I} + \mathbf{\Omega} + \frac{1}{2!}(\mathbf{\Omega})^2 + \frac{1}{3!}(\mathbf{\Omega})^3 + ... \quad . \tag{7}$$

where $\mathbf{I} \in \text{SO}(3)$ is the identity matrix. In addition, $\mathbf{\Omega}$ can be mapped from an element $\boldsymbol{\omega} = (\omega_x, \omega_y, \omega_z)$ in Euclidean space $\mathbb{R}^3$ through a skew-symmetric operator $\Pi_\mathbf{R} : \mathbb{R}^3 \to \mathcal{T}_\mathbf{R}\mathcal{M}$ as

$$\Pi_\mathbf{R}(\boldsymbol{\omega}) = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \tag{8}$$

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

$$\text{Exp}_\mathbf{I}(\Pi_\mathbf{I}(\boldsymbol{\omega})) = \text{Exp}_\mathbf{I}(\Pi_\mathbf{I}(\theta\hat{\boldsymbol{\omega}})) = \mathbf{I} + sin\theta\Pi_\mathbf{I}(\hat{\boldsymbol{\omega}}) + (1 - cos\theta)(\Pi_\mathbf{I}(\hat{\boldsymbol{\omega}}))^2 \tag{9}$$

which is well known as the Rodrigues formula (Rodrigues, 1840). Due to the nature of the Lie group, we can expand the formula in eq. (8) 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 $\mathbf{R}$:

$$\text{Exp}_\mathbf{R}(\Pi_\mathbf{R}(\boldsymbol{\omega})) = \mathbf{R}(\text{Exp}_\mathbf{I}(\Pi_\mathbf{R}(\boldsymbol{\omega}))) = \mathbf{R}(\mathbf{I} + sin\theta\Pi_\mathbf{R}(\hat{\boldsymbol{\omega}}) + (1 - cos\theta)(\Pi_\mathbf{R}(\hat{\boldsymbol{\omega}}))^2)$$

Following (Taylor & Kriegman, 1994), we have

$$\frac{\partial}{\partial\omega_x}\text{Exp}_\mathbf{R}(\Pi_\mathbf{R}(\boldsymbol{\omega}))\Big|_{\boldsymbol{\omega}=\mathbf{0}} = \mathbf{R}\frac{\partial}{\partial\omega_x}\left(\sum_{n=0}^{\infty}(\frac{1}{n!}(\Pi_\mathbf{R}(\boldsymbol{\omega}))^n)\right) = \mathbf{R}\Pi_{\mathbf{R}(\hat{\mathbf{x}})} \tag{10}$$

where $\hat{\mathbf{x}} = (1, 0, 0) \in \mathbb{R}^3$. For ω_y and ω_z, there are the similar expressions of the gradient.

Therefore,

$$\text{grad}f(\mathbf{R}) = \Pi_\mathbf{R}(\nabla f(\mathbf{R})) = \Pi_\mathbf{R}(\nabla\boldsymbol{\omega}) = \Pi_\mathbf{R}\left(\frac{\partial f(\mathbf{R})}{\partial\mathbf{R}}\frac{\partial}{\partial\boldsymbol{\omega}}\text{Exp}_\mathbf{R}(\Pi_\mathbf{R}(\boldsymbol{\omega}))\Big|_{\boldsymbol{\omega}=\mathbf{0}}\right) \tag{11}$$

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:

$$\mathbf{R}_{k+1} = \mathrm{Exp}_{\mathbf{R}_k}(-\tau_k \nabla \boldsymbol{\omega}). \tag{12}$$

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

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

we can get an analytical expression of $\nabla \boldsymbol{\omega} = (\nabla \omega_x, \nabla \omega_y, \nabla \omega_z)$ as follows:

$$\begin{aligned} \nabla \boldsymbol{\omega}_x &= \frac{\partial f(\mathbf{R})}{\partial \mathbf{R}} * \mathbf{R} \Pi_{\mathbf{R}}(\hat{x}) \\ &= 2 \left|\left| \begin{pmatrix} a_1 - x_1 & b_1 - y_1 & c_1 - z_1 \\ a_2 - x_2 & b_2 - y_2 & c_2 - z_2 \\ a_3 - x_3 & b_3 - y_3 & c_3 - z_3 \end{pmatrix} \begin{pmatrix} 0 & c_1 & -b_1 \\ 0 & c_2 & -b_2 \\ 0 & c_3 & -b_3 \end{pmatrix} \right|\right|_1 \\ &= 2 * \sum_{i=1}^{3} (b_i * z_i - c_i * y_i) \end{aligned} \tag{13}$$

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

τ_{safe} in ablation study. We have mentioned in the main paper that τ should neither be too large nor too small. To be more specific, what we want is a small τ at the beginning of training and a large τ when converging. This is because a small τ can yield $\mathbf{R}_g$ closer to $\mathbf{R}$ and greatly alleviate the problem discussed in Section 3.3 at the beginning stage of training. 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 can find a suitable constant value to act like τ_{gt} when converging, which we denotes as τ_{safe}.

Lemma 1. *The final value of τ_{safe} satisfies:*

$$\mathbf{R}_{\mathrm{gt}} = \lim_{<\mathbf{R}, \mathbf{R}_{\mathrm{gt}}> \to 0} R_{\mathbf{R}}(-\tau_{safe} \operatorname{grad} \mathcal{L}(f(\mathbf{R}))) \tag{14}$$

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 $\mathbf{R} = \mathbf{I}$. This will simplify the derivation. Based upon the conclusion in eq. (13), when we use L2 loss, we have $\nabla \boldsymbol{\omega} = (2 * (z_2 - y_3), 2 * (x_3 - z_1), 2 * (y_1 - x_2))$ and $\operatorname{grad} \mathcal{L} f(\mathbf{R}) = \Pi_{\mathbf{R}}(\nabla \boldsymbol{\omega}) = 2(\mathbf{R}_{\mathrm{gt}}^\top - \mathbf{R}_{\mathrm{gt}})$.

Taking the manifold logarithm of both sides, we get:

$$\mathrm{Log}_{\mathbf{R}}(\mathbf{R}_{\mathrm{gt}}) = \lim_{<\mathbf{R}, \mathbf{R}_{\mathrm{gt}}> \to 0} -\tau_{safe} \operatorname{grad} \mathcal{L} f(\mathbf{R}) \tag{15}$$

The solution for τ_{safe} can then be derived as follows:

$$\begin{aligned} \tau_{safe} &= \lim_{<\mathbf{R}, \mathbf{R}_{\mathrm{gt}}> \to 0} -\frac{\mathrm{Log}_{\mathbf{R}}(\mathbf{R}_{\mathrm{gt}})}{\operatorname{grad} \mathcal{L} f(\mathbf{R})} = \lim_{<\mathbf{I}, \mathbf{R}_{\mathrm{gt}}> \to 0} -\frac{\mathrm{Log}_{\mathbf{I}}(\mathbf{R}_{\mathrm{gt}})}{\operatorname{grad} \mathcal{L} f(\mathbf{I})} = \lim_{\theta \to 0} -\frac{\Pi_{\mathbf{I}}(\boldsymbol{\omega}_{\mathrm{gt}})}{2(\mathbf{R}_{\mathrm{gt}}^\top - \mathbf{R}_{\mathrm{gt}})} \\ &= \lim_{\theta \to 0} -\frac{\Pi_{\mathbf{I}}(\boldsymbol{\omega}_{\mathrm{gt}})}{2((\mathbf{I} + sin\theta \Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}})^\top + cos\theta (\Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}})^\top)^2) - (\mathbf{I} + sin\theta \Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}}) + cos\theta (\Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}}))^2))} \\ &= \lim_{\theta \to 0} -\frac{\Pi_{\mathbf{I}}(\boldsymbol{\omega}_{\mathrm{gt}})}{2sin\theta (\Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}})^\top - \Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}}))} \\ &= \lim_{\theta \to 0} \frac{\theta}{4sin\theta} \\ &= \frac{1}{4} \end{aligned} \tag{16}$$

where $\Pi_{\mathbf{I}}(\boldsymbol{\omega}_{\mathrm{gt}}) = \mathrm{Log}_{\mathbf{I}}(\mathbf{R}_{\mathrm{gt}}) = \theta \Pi_{\mathbf{I}}(\hat{\boldsymbol{\omega}}_{\mathrm{gt}})$ and $\theta = < \mathbf{I}, \mathbf{R}_{\mathrm{gt}} >$ □

Note that when we use a $\tau \leq \frac{1}{4}$, $\mathbf{R}_g$ will always be closer than $\mathbf{R}_{\text{gt}}$ to $\mathbf{R}$. So we set $\tau = \frac{1}{4}$ as our upper bound of τ schedule, and call $\frac{1}{4}$ as τ_{safe}. Note that this is only true for the L2 loss.

B.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. Please refer to Figure 1 for this process.

Quaternion We need to solve

$$\mathbf{q}_{gp} = \underset{\mathbf{q}_g \in \pi_q^{-1}(\hat{\mathbf{q}}_g)}{\operatorname{argmin}} ||\mathbf{q} - \mathbf{q}_g||_2^2, \tag{17}$$

where $\mathbf{q}$ is in ambient space and $\hat{\mathbf{q}}_g$ is the next goal in representation manifold. 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

$$||\mathbf{q} - \mathbf{q}_g||_2^2 = \mathbf{q}^2 - 2k\mathbf{q} \cdot \hat{\mathbf{q}}_g + k^2\hat{\mathbf{q}}_g^2 \tag{18}$$

Without considering the condition of $k > 0$, We can see when $k = \frac{\mathbf{q} \cdot \hat{\mathbf{q}}_g}{\hat{\mathbf{q}}_g^2} = \mathbf{q} \cdot \hat{\mathbf{q}}_g$ the target formula reaches minimum. Note that when using a small τ, the angle between $\hat{\mathbf{q}}_g$ and $\mathbf{q}$ is always very small, which means the condition of $k = \mathbf{q} \cdot \hat{\mathbf{q}}_g > 0$ 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 $\mathbf{q}_{gp} = (\mathbf{q} \cdot \hat{\mathbf{q}}_g)\hat{\mathbf{q}}_g$.

6D representation We need to solve

$$[\mathbf{u}_{gp}, \mathbf{v}_{gp}] = \underset{[\mathbf{u}_g, \mathbf{v}_g] \in \pi_{6D}^{-1}([\hat{\mathbf{u}}_g, \hat{\mathbf{v}}_g])}{\operatorname{argmin}} (||\mathbf{u} - \mathbf{u}_g||_2^2 + ||\mathbf{v} - \mathbf{v}_g||_2^2) \tag{19}$$

where $[\mathbf{u}, \mathbf{v}]$ is in ambient space and $[\hat{\mathbf{u}}_g, \hat{\mathbf{v}}_g]$ is the next goal in representation manifold. 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:

$$||\mathbf{v} - \mathbf{v}_g||_2^2 = \mathbf{v}^2 + k_2^2\hat{\mathbf{u}}_g^2 + k_3^2\hat{\mathbf{v}}_g^2 - 2k_2\mathbf{v} \cdot \hat{\mathbf{u}}_g - 2k_3\mathbf{v} \cdot \hat{\mathbf{v}}_g \tag{20}$$

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, the situation is quite different from the above two and obtaining the inverse image π_{9D}^{-1} is not so obvious. Recall $\pi_{9D}(\mathbf{M}) = \mathbf{U}\Sigma'\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ are left and right singular vectors of $\mathbf{M}$ decomposed by SVD expressed as $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$, and $\Sigma' = \mathbf{d}(1, 1, \det(\mathbf{U}\mathbf{V}^\top))$. To find a suitable π_{9D}^{-1}, the most straightforward way is to only change the singular values $\Sigma_g = \mathbf{d}(\lambda_0, \lambda_1, \lambda_2)$, where $\lambda_0, \lambda_1, \lambda_2$ can be arbitrary scalars, and recompose the $\mathbf{M}_g = \mathbf{U}\Sigma_g\mathbf{V}^\top$. However, we argue that this simple method will fail to capture the entire set of $\{\mathbf{M}_g\}$ that would satisfy $\pi_{9D}(\mathbf{M}_g) = \mathbf{R}_g \in \text{SO}(3)$. This is because different $\mathbf{U}'$ and $\mathbf{V}'$ can yield the same rotation $\mathbf{R}_g$. In fact, $\mathbf{U}_g$ can be arbitrary if $\mathbf{M}_g = \mathbf{U}_g\Sigma_g\mathbf{V}_g^\top$ 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 $\mathbf{M}_g$ in a different way: $\mathbf{M}_g = \mathbf{U}_g\Sigma_g\frac{1}{\Sigma_g'}\mathbf{U}_g^{-1}\mathbf{R}_g$. 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, we get $\pi_{9D}^{-1}(\mathbf{R}_g) = \{\mathbf{S}\mathbf{R}_g \mid \mathbf{S} \text{ is an arbitrary symmetric matrix}\}$. Note that such $\mathbf{M}_g \in \pi_{9D}^{-1}(\mathbf{R}_g)$ can't ensure $\pi_{9D}(\mathbf{M}_g) = \mathbf{R}_g$, because in the implementation of SVD, the order and the sign of three singular values are constrained, which is not taken into consideration.

We need to solve

$$\mathbf{M}_{gp} = \underset{\mathbf{M}_g \in \pi_{9D}^{-1}(\mathbf{R}_g)}{\operatorname{argmin}} (||\mathbf{M} - \mathbf{M}_g||_2^2) \tag{21}$$

We can further transform this optimization objective as follows

$$
\begin{aligned}
||\mathbf{M}-\mathbf{M}_g||_2^2 = ||\mathbf{M}-\mathbf{S}\mathbf{R}_g||_2^2 = ||\mathbf{M}\mathbf{R}_g^\top-\mathbf{S}||_2^2 &= \sum_{i=1}^{3}\sum_{j=1}^{3}(m_{ij}-s_{ij})^2 \\
&= \sum_{i=1}^{3}\sum_{j=1}^{i-1}((m_{ij}-s_{ij})^2+(m_{ji}-s_{ij})^2)+\sum_{i=1}^{3}(m_{ii}-s_{ii})^2 \\
&= \sum_{i=1}^{3}\sum_{j=1}^{i-1}(2s_{ij}^2-2s_{ij}(m_{ji}+m_{ij})+m_{ij}^2+m_{ji}^2)+\sum_{i=1}^{3}(m_{ii}-s_{ii})^2 \qquad (22)
\end{aligned}
$$

where $\mathbf{S}=(s_{ij})_{i,j=1,2,3}$ and $\mathbf{M}\mathbf{R}_g^\top=(m_{ij})_{i,j=1,2,3}$.

Now we can easily find when $s_{ij}=\begin{cases}\frac{m_{ij}+m_{ji}}{2} & (i\neq j)\\ m_{ii} & (i=j)\end{cases}$, in other words, when $\mathbf{S}$ equals to the symmetry part of $\mathbf{M}\mathbf{R}_g^\top$, the target formula reaches minimum. Therefore, the inverse projection admits a simple form $\mathbf{M}_{gp}=\frac{\mathbf{M}\mathbf{R}_g^\top+\mathbf{R}_g\mathbf{M}^\top}{2}\mathbf{R}_g$.

C Projective Manifold Gradient on $\mathcal{S}^2$

C.1 Riemannian optimization on $\mathcal{S}^2$

Our methods can also be applied for the regression of other manifolds. Taking $\mathcal{S}^2$ as an example, which is included in the experiment part 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 $\mathrm{Exp}_{\hat{\mathbf{x}}}(\mathbf{v})=cos(||\mathbf{v}||)\mathbf{x}+sin(||\mathbf{v}||)\frac{\mathbf{v}}{||\mathbf{v}||}$, where $||.||$ means the ordinal Frobenius norm.

For the definition of the mapping $\Pi_{\hat{\mathbf{x}}}$, 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 $\Pi_{\hat{\mathbf{x}}}(\boldsymbol{\omega})=\omega_1\hat{\mathbf{c}}_1+\omega_2\hat{\mathbf{c}}_2$, where $\boldsymbol{\omega}=(\omega_1,\omega_2)\in\mathbb{R}^2$. The gradient of exponential mapping with respect to $\boldsymbol{\omega}$ is

$$
\frac{\partial}{\partial\omega_1}\mathrm{Exp}_{\hat{\mathbf{x}}}(\Pi_{\hat{\mathbf{x}}}(\boldsymbol{\omega}))\Big|_{\boldsymbol{\omega}=\mathbf{0}} = \frac{\partial}{\partial\omega_1}(cos(||\omega_1\hat{\mathbf{c}}_1||)\mathbf{x}+sin(||\omega_1\hat{\mathbf{c}}_1||)\frac{\omega_1\hat{\mathbf{c}}_1}{||\omega_1\hat{\mathbf{c}}_1||})\Big|_{\boldsymbol{\omega}=\mathbf{0}}=\hat{\mathbf{c}}_1 \qquad (23)
$$

Similarly, we have $\frac{\partial}{\partial\omega_2}\mathrm{Exp}_{\hat{\mathbf{x}}}(\Pi_{\hat{\mathbf{x}}}(\boldsymbol{\omega}))\Big|_{\boldsymbol{\omega}=\mathbf{0}}=\hat{\mathbf{c}}_2$.

When using L2 loss, we can have

$$
\begin{aligned}
\mathrm{grad}f(\hat{\mathbf{x}}) = \Pi_{\hat{\mathbf{x}}}(\nabla f(\hat{\mathbf{x}})) = \Pi_{\hat{\mathbf{x}}}(\nabla\boldsymbol{\omega}) &= \Pi_{\hat{\mathbf{x}}}\left(\frac{\partial f(\hat{\mathbf{x}})}{\partial\hat{\mathbf{x}}}\frac{\partial}{\partial\boldsymbol{\omega}}\mathrm{Exp}_{\hat{\mathbf{x}}}(\Pi_{\hat{\mathbf{x}}}(\boldsymbol{\omega}))\Big|_{\boldsymbol{\omega}=\mathbf{0}}\right) \\
&= \Pi_{\hat{\mathbf{x}}}((2(\hat{\mathbf{x}}-\hat{\mathbf{x}}_{gt})\hat{\mathbf{c}}_1, 2(\hat{\mathbf{x}}-\hat{\mathbf{x}}_{gt})\hat{\mathbf{c}}_2)) \\
&= \Pi_{\hat{\mathbf{x}}}((-2||(\hat{\mathbf{x}}\cdot\hat{\mathbf{x}}_{gt})\hat{\mathbf{x}}-\hat{\mathbf{x}}_{gt}||, 0)) \\
&= 2((\hat{\mathbf{x}}\cdot\hat{\mathbf{x}}_{gt})\hat{\mathbf{x}}-\hat{\mathbf{x}}_{gt}) \qquad (24)
\end{aligned}
$$

Similar to Eq 16, we can also solve a τ_{safe}

$$\tau_{safe} = \lim_{<\hat{\mathbf{x}},\hat{\mathbf{x}}_{\text{gt}}>\to 0} -\frac{\text{Log}_{\hat{\mathbf{x}}}(\hat{\mathbf{x}}_{\text{gt}})}{\text{grad}\ \mathcal{L}f(\hat{\mathbf{x}})} = \lim_{\theta\to 0} \frac{\theta\hat{\mathbf{c}}_1}{2sin\theta\hat{\mathbf{c}}_1} = \frac{1}{2} \tag{25}$$

where $\theta =< \hat{\mathbf{x}}, \hat{\mathbf{x}}_{gt} >$.

Note that in Experiment 4.4, we change the schedule of τ according to this conclusion. We increase τ from 0.1 to 0.5 by uniform steps.

C.2 Inverse projection

It is exactly the same as quaternion. We can have $\mathbf{x}_{gp} = (\mathbf{x} \cdot \hat{\mathbf{x}}_g)\hat{\mathbf{x}}_g$. For the detail of derivation, see Sec B.2.

D More experiments

D.1 3D object pose estimation from ModelNet image dataset

In this experiment, we follow the setting in (Levinson et al., 2020) to estimate poses from 2D images. Images are rendered from ModelNet-10 (Wu et al., 2015) objects from arbitrary viewpoints (Liao et al., 2019). A MobileNet (Howard et al., 2017) is used to extract image features and three MLPs to regress rotations. Similarly, we focus on *chair* and *sofa* categories which exhibit the least rotational symmetries in the dataset. Note that we conduct all the experiments by our code rather than quote the numbers to ensure a fair comparison.

The results are shown in Table 7 and Table 8. Clearly, our RPMG layer boosts the performance of all three representations significantly. See the curves with the same color for comparison.

	Mean (°)	Med (°)	5°Acc (%)
Euler	21.46	10.95	10.4
Axis-Angle	25.71	14.27	7.2
Quaternion	25.75	14.99	6.3
6D	19.60	9.09	19.1
9D	17.46	8.30	23.1
9D-Inf	12.10	5.09	49.2
10D	18.40	9.02	19.6
RPMG-Quat	13.03	5.90	39.9
RPMG-6D	12.94	4.74	53.1
RPMG-9D	**11.93**	**4.36**	**58.1**

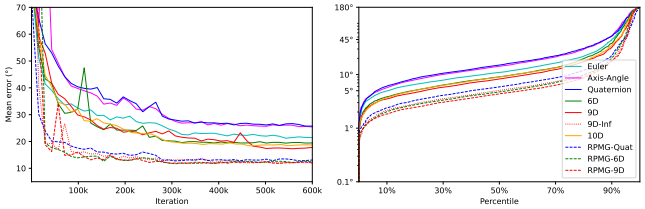

Table 7: **Pose estimation from ModelNet *chair* images**. We report the same metrics as in Table 1; see the caption there. All models are trained for 600K iterations.

	Mean (°)	Med (°)	5°Acc (%)
Euler	27.46	12.00	9.4
Axis-Angle	30.25	14.55	6.2
Quaternion	30.00	15.73	5.7
6D	17.51	7.33	27.3
9D	19.75	7.58	24.9
9D-Inf	12.48	3.45	69.7
10D	20.89	8.73	19.8
RPMG-Quat	13.02	3.60	66.6
RPMG-6D	11.52	2.79	77.1
RPMG-9D	**10.49**	**2.41**	**81.7**

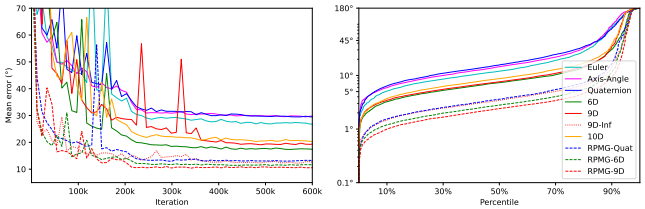

Table 8: **Pose estimation from ModelNet *sofa* images.** We report the same metrics as in Table 1; see the caption there. All models are trained for 600K iterations.

D.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 if many computer vision and robotic applications. In this experiment, we use all the settings (data, network, training strategy, hyperparameters, etc.) of PoseLSTM (Walch et al., 2017) except that we modify the rotation representations and the gradient layers. We report the results on the outdoor Cambridge Landscape dataset (Alex Kendall & Cipolla, 2015).

	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**

Table 9: **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.

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.

D.3 Regression on Other Non-Euclidean Manifolds

In addition to SO(3), our method can also be applied for regression on other non-Euclidean manifolds as long as the target manifold meets some conditions: 1. the manifold should support Riemannian optimization. 2. the inverse projection π^{-1} should be calculable, although it doesn't need to be mathematically complete. Here we show the experiment of *Sphere manifold* $\mathcal{S}^2$.

Unit vector regression For rotational symmetric categories (e.g., *bottle*), the pose of an object is ambiguous. We'd rather regress a unit vector for each object indicating the *up* direction of it. We use the ModelNet(Wu et al., 2015) *bottle* point cloud dataset. The network architecture is the same as in Sec4.1 except the dimension of output is 3.

L2-loss-w/-norm computes L2 loss between the normalized predictions and the ground truth. L2-loss-w/o-norm computes L2 loss between the raw predictions and the ground truth. MG-3D and PMG-3D are two variants of our method. See Sec4.1 for details.

Table 10 shows the geodesic error statistics. MG-3D performs on par with L2-loss-w/o-norm, and PMG-3D leads to a large error since the norm of the predicted vectors is extremely small (10^3) without the regularization term. RPMG-3D outperforms all the baselines and variants.

	Complete			Depth		
	Mean ($^\circ$)	Med ($^\circ$)	1°Acc (%)	Mean ($^\circ$)	Med ($^\circ$)	1°Acc (%)
L2 loss w/ norm	8.73	2.71	0.0	6.00	4.41	0.0
L2 loss w/o norm	5.71	1.10	37.4	3.25	1.69	3.0
MG-3D	5.37	1.20	22.2	2.94	2.01	0.0
PMG-3D	21.96	14.79	0.0	32.35	22.56	0.0
RPMG-3D	**4.69**	**0.76**	**72.7**	**2.16**	**1.37**	**13.1**

Table 10: **Unit vector estimation from ModelNet bottle *complete* and *depth* point clouds.** We report the same metrics as in Table 1 except replace 5°Acc by 1°Acc; see the caption there. All models are trained for 30K iterations.

E More Implementation Details

E.1 Experiment 4.1 & 4.3 & 4.4

Data We generate the data from ModelNet dataset (Wu et al., 2015), following the same generation method as in (Zhou et al., 2019). For complete point clouds, 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. For depth point clouds, we render the rotated complete point clouds to depth images and back-project to partial point clouds.

Network Architecture We use a PointNet++ MSG (Qi et al., 2017) 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.

The learning rate is set to 1e-3 and decayed by 0.7 every 3k iterations. The batch size is 10. For each experiment, we train the network on one NVIDIA TITAN Xp GPU for 5-7 hours.

F STANDARD MAPPING BETWEEN ROTATION MATRIX AND QUATERNION

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

$$\phi(\mathbf{q}) = \begin{pmatrix} 2(q_0^2 + q_1^2) - 1 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & 2(q_0^2 + q_2^2) - 1 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & 2(q_0^2 + q_3^2) - 1 \end{pmatrix} \tag{26}$$

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:

$$\begin{cases} q_0 = \sqrt{1 + R_{00} + R_{11} + R_{22}} \\ q_1 = (R21 - R_{12})/(4 * q_0) \\ q_2 = (R_{02} - R_{20})/(4 * q_0) \\ q_3 = (R_{10} - R_{01})/(4 * q_0) \end{cases} \tag{27}$$

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 $\mathbf{R}$.