Learning with 3D Rotations: a Hitchhiker's Guide to SO(3)

A. René Geist [1 2] **Jonas Frey** [1 3] **Mikel Zhobro** [1 2] **Anna Levina** [4] **Georg Martius** [1 2]

Abstract

Many settings in machine learning require the selection of a rotation representation. However, choosing a suitable representation from the many available options is challenging. This paper acts as a survey and guide through rotation representations. We walk through their properties that harm or benefit deep learning with gradient-based optimization. By consolidating insights from rotation-based learning, we provide a comprehensive overview of learning functions with rotation representations. We provide guidance on selecting representations based on whether rotations are in the model's input or output and whether the data primarily comprises small angles.

The project code is available at:
github.com/martius-lab/hitchhiking-rotations

1. Introduction

For centuries, researchers explored methods to describe the rotation of Cartesian coordinate systems in three-dimensional Euclidean space. Euler (1765) demonstrated the necessity of at least three parameters for parameterizing 3D rotations through rotation matrices $R \in \mathrm{SO}(3)$ as exemplified by Euler angles. Subsequent discoveries led to the description of rotation through exponential coordinates and quaternions, which found widespread adoption in many fields such as control theory, robotics, and computer animations. The choice of a rotation representation for a specific application depends on its properties, including human-interpretability, computational costs, and dimensionality. In recent years, the representation's impact on learning functions from data has become another significant factor.

[1]Max Planck Institute for Intelligent Systems, Autonomous learning group, Germany [2]University of Tübingen, Distributed intelligence lab, Germany [3]Swiss Federal Institute of Technology (ETH Zurich), Robotic systems lab, Switzerland [4]University of Tübingen, Self-organization of neuronal networks group, Germany. Correspondence to: A. René Geist <andreas-rene.geist@wsii.uni-tuebingen.de>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

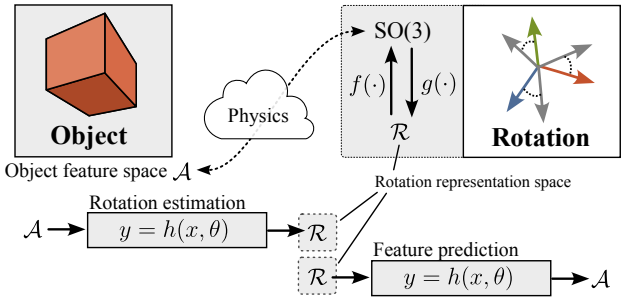

Figure 1. Overview on learning with rotations. A neural network learns a function from a feature space $\mathcal{A}$ to a rotation representation space $\mathcal{R}$ or vice versa. When learning with rotations, the properties of $f : \mathcal{R} \to \mathrm{SO}(3)$ and $g : \mathrm{SO}(3) \to \mathcal{R}$ affect training.

In machine learning, we want to choose the representation of our data that yields the best test accuracy. However, recent works (Zhou et al., 2019; Brégier, 2021; Levinson et al., 2020) suggest that rotation representations with four or less dimensions do not facilitate sample-efficient learning. To further complicate matters, empirical findings by Pepe et al. (2022) suggest that geometric algebra representations may rival the high-dimensional representations proposed by Zhou et al. (2019); Levinson et al. (2020). Despite many recent works discussing the role of rotations in machine learning, we did not encounter a work that provides a comprehensive overview on the question:

What representation of $\mathrm{SO}(3)$ *is suitable for neural network regression with gradient-based optimization?*

Besides considering the case in which rotations are in the models' output as discussed in Zhou et al. (2019); Brégier (2021); Levinson et al. (2020), we also detail how one may learn functions from rotations being in the input. We discuss common rotation representations, using tricks such as wrapping angle coordinates into sinusoidal transformations, problems arising from double-cover, and how the choice of metric affects training. We find that high-dimensional representations should be the preferred default choice from a theoretical as well as empirical perspective. As it turns out, many aspects of a rotation representation influence training while also vice versa, the setup of machine learning problem influences the choice of rotation representation.

Table 1. Overview of selected rotation representations with some of their properties. Due to singularities arising from angle coordinates and double cover (explained below), rotation representations with three and four dimensions have a discontinuous map $g(R) : \mathrm{SO}(3) \to \mathcal{R}$.

2D/3D	Representation	Notation	Dim	Domain	$g(R)$ cont.	Uses Angles	Double cover
SO(2)	Angle	α	1	$\mathbb{R}^1$	-	yes	no
	Sine and cosine of the angle	$[\cos(\alpha), \sin(\alpha)]$	2	$\mathbb{R}^2$	+	no	no
SO(3)	Euler parameters	Euler	3	$\mathbb{R}^3$	-	yes	yes
	Exponential coordinates / Bi-Vectors	Exp	3	$\mathbb{R}^3$	-	no	yes
	Unit-quaternions / Rotors	Quat	4	S^3	-	no	yes
	Axis-angle	Axis-angle	4	$\mathbb{R}^4$	-	yes	yes
	$\mathbb{R}^6$+Gram-Schmidt orthonormalization	$\mathbb{R}^6$+GSO	6	$\mathbb{R}^{3\times 2}$	+	no	no
	$\mathbb{R}^9$+singular value decomposition	$\mathbb{R}^9$+SVD	9	$\mathbb{R}^{3\times 3}$	+	no	no

1.1. Problem setting

In this work, we consider gradient-based supervised neural network regression. That is, given a data set $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^N$ of N inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$, find the parameters θ of the neural network $h : \mathcal{X} \to \mathcal{Y}$

$$y = h(x; \theta) \tag{1}$$

that minimize the loss function

$$L(\mathcal{D}, \theta) = \sum_{x,y \in \mathcal{D}} d(y, h(x, \theta)) \tag{2}$$

using the parameter gradient $\nabla_\theta L$. The loss measures the average distances d of predictions to target values. For simplicity, we assume that there exists a deterministic target function $y = h^*(x)$ that generated the data.

Representing rotations To work with rotations, they have to be suitably represented. In analogy to Zhou et al. (2019), we say a vector $r \in \mathcal{R} \subseteq \mathbb{R}^d$ is a rotation representation, if there exist two functions $f : \mathcal{R} \to \mathrm{SO}(N)$ and $g : \mathrm{SO}(N) \to \mathcal{R}$ such that $f(g(R)) = R$ that is f is a left inverse of g. Note that depending on the choice of the representation the topology and dimensionality d of $\mathcal{R}$ may vary. Table 1 lits common representations that differ in their dimensionality and topology.

Learning scenarios To analyze the consequences of different representations on learning, we consider the cases where rotations occur in the input ($\mathcal{R} \subseteq \mathcal{X}$) and where they occur in the output ($\mathcal{R} \subseteq \mathcal{Y}$) of our regression problem. As depicted in Figure 1, we denote the features of an entity of interest by features $a \in \mathcal{A}$. This could be, for instance, a camera image, a point cloud, or adhesive forces acting between two molecules. For simplicity, we ignore possible other components and consider the two pure cases of:

i) *feature prediction* where $a = h(r)$ with $\mathcal{X} = \mathcal{R}$ and $\mathcal{Y} = \mathcal{A}$,
ii) *rotation estimation* where $r = h(a)$ with $\mathcal{X} = \mathcal{A}$ and $\mathcal{Y} = \mathcal{R}$,

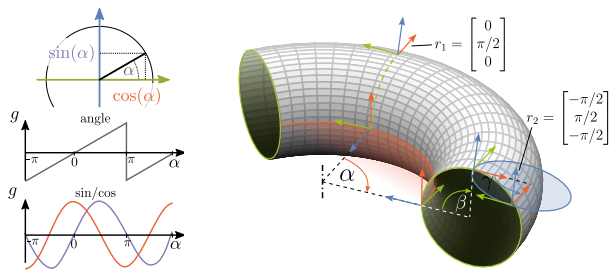

Figure 2. Left: Representations for SO(2): angle and sin/cos with respective g functions. Right: Euler angles representation for SO(3). In Euler angles a frame can be visualized as three consecutive SO(2) rotations along the surface of a torus.

In feature prediction, we are interested in learning a function from r (and other object variables) to a particular object property a, for example, rendering an object from a particular direction or predicting dynamics. In rotation estimation, a map from a high-dimensional representation a to r is learned, for example, pose estimation from images.

2. Representations of rotation

In this section, we describe common rotation representations as detailed in Table 1 and examine their properties. An overview is provided in Table 1.

SO(2) representations Rotation angles seem to be the most natural representation. A 2D rotation is fully described by a single angle α. However, because the angle is a 1D quantity, there is a jump in the function g, as shown on the left in Figure 2. If the rotation is represented by $(\cos(\alpha), \sin(\alpha))$, such a discontinuity does not arise.

Euler angles In 3D we need at least three angles $\alpha, \beta, \gamma \in [-\pi, \pi)$ (Euler angles) to describe a rotation. Rotation matrices can be composed through a sequence of elementary rotations $R(\alpha, \beta, \gamma) = R_3(\gamma)R_2(\beta)R_1(\alpha) := f(r)$ by these angles. Discontinuities arise when bounding the range of angles (see above). In addition, for Euler angles, the same

point in $\mathrm{SO}(3)$ can be described by different representation vectors. For example, in Figure 2 the points $r_1 = [0, \pi/2, 0]$ and $r_2 = [-\pi/2, \pi/2, -\pi/2]$ perform the same rotation.

Due to these and other reasons, studies on learning with rotations, including Huynh (2009); Zhou et al. (2019); Brégier (2021); Pepe et al. (2022), uniformly discourage the use of Euler angles in learning with 3D rotations.

Exponential coordinates Each rotation in 3D can also be expressed by a rotation axis $\omega \in \mathbb{R}^3$ and an angle. One possible representation is thus to use the length of the vector $\|w\|$ to encode the angle. The identity rotation corresponds to the origin of $\mathbb{R}^3$ (zero vector) and spheres of radius $2n\pi$ with $n \in \mathbb{N}^+$. This representation is called *exponential coordinates* because ω can be written as a 3×3 skew-symmetric matrix. The matrix exponential of that yields the desired rotation matrix. Alternative names are "rotation vectors", "Rodrigues parameters", or "angular velocity". Figure 3 shows the concept. It also illustrates that the same rotation can be expressed by two vectors (inside the sphere of radius 2π) as $f(\omega) = f\big((\|\omega\| - 2\pi)\frac{\omega}{\|\omega\|}\big)$ (double cover).

Axis angle and Quaternions Instead of tying the length of the axis vector to the rotation, we can separately represent them as $r = (\tilde{\omega}, \alpha) \in \mathbb{R}^4$ with $\|\tilde{\omega}\| = 1$, logically denoted as *axis angle*. Figure 3 illustrates the same double cover property as exponential coordinates. We can transform exponential coordinates to axis angles and then the axis angle vector to $\mathrm{SO}(3)$ via Rodrigues' rotation formula $f(r) := \mathbb{I} + \sin(\alpha)[\tilde{\omega}]_\times + (1 - \cos(\alpha))[\tilde{\omega}]_\times)^2$ whereas the map from $\mathrm{SO}(3)$ to exponential coordinates is given by the matrix exponential $g(R) := \exp(R)$.

Similar representations are *quaternions*, which extend the concept of complex numbers to higher dimensions. For the purpose of rotations, we consider unit quaternions that relate to the axis angle representation $(\tilde{\omega}, \alpha)$ as $r = q = (w, x, y, z) \in \mathcal{S}^3$, i.e. $\|q\| = 1$ and $w = \cos(\alpha/2)$ and $(x, y, z) = \sin(\alpha/2)\tilde{\omega}$ (Grassia, 1998). Unit quaternions double cover $\mathrm{SO}(3)$ such that $f(q) = f(-q)$.

$\mathbb{R}^6$ + Gram-Schmidt orthonormalization (GSO) A representation that is closer to the actual rotation matrix is $r = (\nu_1, \nu_2) \in \mathbb{R}^{3\times 2}$. A rotation matrix can be obtained by using Gram-Schmidt orthonormalization (GSO) to "complete" a Cartesian frame (Zhou et al., 2019). As the columns of a rotation matrix are unit length and orthogonal to each other, this method yields a rotation matrix $R = f(r) = \mathrm{GSO}(\nu_1, \nu_2)$ whereas $g(R) := \mathrm{diag}(1, 1, 0)R$. The simple idea of GSO is to first normalize ν_1, subtract from ν_2 its component that is colinear with ν_1 to obtain $\nu_2^\perp$, then after normalizing $\nu_2^\perp$, we obtain ν_3 as the cross product of ν_1 and $\nu_2^\perp$. GSO straightforwardly extends to $\mathrm{SO}(n)$ (Macdonald, 2010, p. 120).

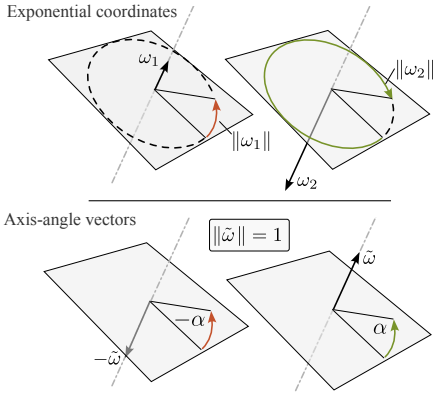

Figure 3. Exponential coordinates (Exp) and axis-angle representation and their double cover property. **Top**: Exp. coord.: rotation around ω by angle $\|\omega\|$. The vector $\omega_1 = \alpha_1\tilde{\omega}_1 \in \mathbb{R}^3$ describes the same rotation as $\omega_2 = (\|\omega_1\| - 2\pi)\omega_1/\|\omega_1\|$. **Bottom:** Axis-angles explicitly represent axis and angle: $\tilde{\omega} \in \mathcal{S}^2$, $\alpha \in \mathbb{R}$. The vector $[\tilde{\omega}, \alpha]$ describes the same rotation as $[-\tilde{\omega}, -\alpha]$.

$\mathbb{R}^9$ + singular value decomposition (SVD) Rotations can be directly parameterized by a 3×3 matrix which is then projected to $SO(3)$ using the singular value decomposition (SVD). Given $r = M \in \mathbb{R}^{3\times 3}$ SVD decomposes the matrix into $M = U\Sigma V^T$ where $U, V \in \mathbb{R}^{3\times 3}$ are rotations or reflections and $\Sigma = \mathrm{diag}(\sigma_1, \sigma_2, \sigma_3)$ is a diagonal matrix with the singular values σ_i denoting scaling parameters. Now, the projection is achieved by

$$f(r) := \mathrm{SVD}^+(M) = U\,\mathrm{diag}(1, 1, \det(UV^T))\,V^T, \quad (3)$$

where $\det(UV^T)$ ensures that $\det(\mathrm{SVD}^+(M)) = 1$ (Levinson et al., 2020). As we shall see later, this operation finds the rotation matrix with the least-squares distance to M. The function from $\mathrm{SO}(3)$ to $\mathcal{R}$ is simply $g(R) := R$.

3. Measuring distances between rotations

Supervised regression requires us to measure distances in $\mathcal{Y}$ (Equation (2)). A proper distance metric $d(y_1, y_2)$ is nonnegative, only zero if $y_1 = y_2$ (identity), symmetric, and satisfies the triangular inequality: $d(y_1, y_2) \leq d(y_1, y_3) + d(y_3, y_1)$, yet, when employed as a loss function in machine learning some of these properties can be lifted. When it comes to the estimation of rotation, we need to measure the distances either in $SO(3)$ or in $\mathcal{R}$. In the following discussion, we give a brief overview and geometric understanding of the frequently encountered metrics in rotation learning. A comprehensive overview of the mathematical properties of those metrics is given in Huynh (2009); Hartley et al. (2013); Alvarez-Tunon et al. (2023).

Metrics on Euclidean vector representations We start with general measures (metric and non-metric) that are used to evaluate the similarity between vectors. The Euclidean distance measures the length of the difference vec-

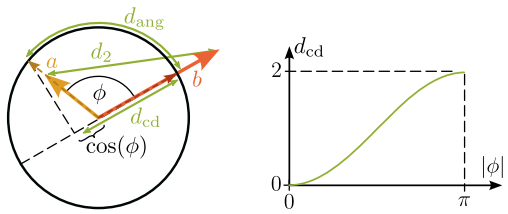

Figure 4. Geometric illustration of distance metrics $d(a, b)$ between vectors a and b. Cosine distance (d_{cd}) and angular distance (d_{ang}) ignore the vectors' lengths.

tor $d_2(a, b) = \|a - b\|_2$, where the L^2 norm of a reads $\|a\|_2 := \|a\| = \sqrt{a \cdot a}$. Using $\cos(\phi) = \frac{a \cdot b}{\|a\|\|b\|}$, we can define the cosine distance and angular distances as:

$$d_{cd}(a, b) = 1 - \cos(\phi), \text{ and } d_{\text{ang}} = \arccos(\cos(\phi)). \quad (4)$$

The angular distance ($\in [0, \pi]$) measures the geodesic distance on a sphere $\mathcal{S}^n$, the cosine distance ($\in [0, 2]$) measures a projection distance of the two (normalized) vectors (Figure 4). Note that cosine distance and angular distance are pseudo-metrics as they violate identity (ignoring the length of the vectors), and cosine distance does not satisfy the triangular inequality, which does not cause problems in practice. Both metrics yield the same ordering of vectors, i.e., for any set of vectors $v_1, \ldots, v_M$, the ordering from closest to furthest from vector u would be the same for both measures.

Metrics that pick distances In Sec. 2, we encounter the problem of double cover, i.e. the same rotation in $SO(3)$ is expressed by two different points in the representation space (Figure 2). For the case of quaternions, we have $f(q) = f(-q)$ such that $d_2(q, -q) \neq 0$ even though the corresponding rotation matrices are identical. An attempt to circumvent this problem is to pick the shortest metric between the quaternions and their negative complement:

$$d_{q,I} = \min(\|q_1 - q_2\|, \|q_1 + q_2\|), \text{ or} \quad (5)$$

$$d_{q,II} = 1 - |q_1 \cdot q_2|. \quad (6)$$

For Euler angles $\alpha, \beta, \gamma \in \mathbb{R}$, Huynh (2009) proposed

$$d_{\text{e}}(r_1, r_2) = \sqrt{d(\alpha_1, \alpha_2)^2 + d(\beta_1, \beta_2)^2 + d(\gamma_1, \gamma_2)^2} \quad (7)$$

with $d(a, b) = \min(|a - b|, 2\pi - |a - b|)$ and $\beta \in [-\pi/2, \pi/2]$ to avoid that two sets of Euler angles represent the same element of SO(3). Note that (5), (6), and (7) are pseudo-metrics on $\mathcal{R}$ but act as metrics on SO(3).

Metrics on rotation matrices The distance between rotation matrices is typically measured using the Frobenius norm, also called Schurnorm. The matrix Frobenius norm of a square matrix $R = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^{n \times n}$ reads

$$\|R\|_{\text{F}} = \sqrt{\sum_i^n \sum_j^n R_{i,j}^2} = \sqrt{\sum_i^n \|v_i\|^2} = \|\text{vec}(R)\|. \quad (8)$$

Recall that the columns of a rotation matrix v_i are the basis vectors of the rotated coordinate frame. By inserting the difference between two rotations into the Frobenius norm (8) one obtains the Chordal distance as

$$d_c(R_1, R_2) = \|R_1 - R_2\|_{\text{F}} = \sqrt{\sum_i^n \|v_{1,i} - v_{2,i}\|^2}, \quad (9)$$

which is an often used metric due to its numerical stability and computational efficiency. For $R_1, R_2 \in \text{SO}(3)$ we have $d_c(R_1, R_2) \in [0, 2\sqrt{2}]$. Due to advantageous convexity properties, the squared Chordal distance $d_c(R_1, R_2)^2$ is often used in practice (Hartley et al., 2013). In turn, the mean squared error between two set of rotations $\{R_{1,1}, ..., R_{1,N}\}$ and $\{R_{2,1}, ..., R_{2,N}\}$ reads $MSE = \sum_{i=1}^N d_c(R_{1,i}, R_{2,i})^2/N = \sum_{i=1}^N \|\text{vec}(R_{1,i} - R_{2,i})\|^2/N$. The geodesic distance can be obtained using multiplication $R_1 R_2^\top$ (which would be I if $R_1 = R_2$) as

$$d_\phi(R_1, R_2) = \arccos \frac{\text{tr}(R_1 R_2^T) - 1}{2}. \quad (10)$$

Figure 18 in the appendix shows the loss function gradients of several of the discussed functions in $\mathbb{R}^2$. Depending on which loss is used the gradients are oriented differently relatively to the unit circle $\mathcal{S}^1$.

4. Rotation representations affect learnability

We analyze now how rotation representations affect the properties of the function $h^*(x)$ which we want to learn with Equation (1). For that, consider the true mapping $\tilde{h}$ from the feature space $\mathcal{A}$ to SO(3) or vice versa. An interesting observation is depicted in Figure 5: the target function h^* is actually the composition of the functions:

(i) For rotation estimation, $h^* = g \circ \tilde{h}$;
(ii) For feature prediction, $h^* = \tilde{h} \circ f$.

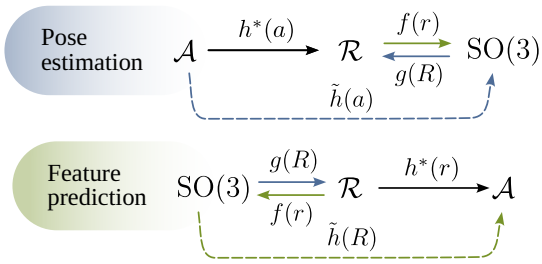

Figure 5. The target function $h^*(x)$ is the composition between $\tilde{h}(x)$ and the functions $g(R)$ / $f(r)$.

With this realization, it is evident that for rotation estimation (i) discontinuities in g or double representations can translate to h^*. For successful gradient-based learning, however, we want h^* to exhibit some notion of **continuity**, as those

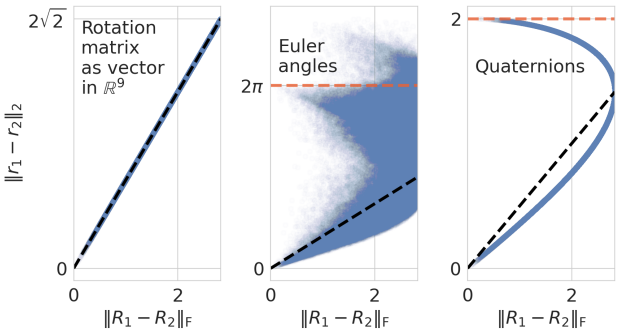

Figure 6. Chordal distance between randomly sampled $R_1, R_2 \in \mathrm{SO}(3)$ and L^2 norm between corresponding rotation representations $r_1 = g(R_1), r_2 = g(R_2)$. The full width of $\mathcal{R}$ is marked by red lines. Ideally, the ratios between the distances, aka the Lipschitz constant of $g(R)$, is close to the black lines whose slope amounts to the ratio between the minimum width of $\mathcal{R}$ and $\mathrm{SO}(3)$.

functions are easier to learn (Xu & Cao, 2004; 2005; Llanas et al., 2008). The *pre-images connectivity constraint* on f, guarantees that there always exists a smooth function that allows to properly interpolate between training representations (Brégier, 2021). For feature prediction (ii), f is continuous for all representations and does not pose a problem with regards to continuity. Albeit, we observe a problem from **disconnectedness**, which we discuss below.

4.1. Why does rotation estimation need high-dimensional rotation parameterization?

To quantify the continuity of a function, we use the concept of Lipschitz continuity that bounds the maximal slope to a constant $\mathcal{L}$ (which would also lead to continuity and require that infinitesimal changes in the function's input result in infinitesimal changes in its output). The mapping function $g(R)$ is Lipschitz continuous if there exist $\mathcal{L} \geq 0$ such that for any $R_1, R_2 \in \mathrm{SO}(3)$ and $r_1 = g(R_1), r_2 = g(R_2), r_1, r_2 \in \mathcal{R}$,

$$\frac{\|r_1 - r_2\|_2}{\|R_1 - R_2\|_{\mathrm{F}}} \leq \mathcal{L}. \tag{11}$$

Discontinuity of g for low-dimensional representations. $SO(3)$ is not homeomorphic to any subset of 4D Euclidean space; thus, for rotation representations with four or fewer dimensions $g(R)$ must be discontinuous (therefore, not Lipschitz continuous) (Stuelpnagel, 1964; Levinson et al., 2020). Figure 6 illustrates these discontinuities by randomly sampling $R_1, R_2 \in \mathrm{SO}(3)$ and then mapping these representations to $r_1 = g(R_1)$ and $r_2 = g(R_2)$. For low-dimensional representations, small distances in $\mathrm{SO}(3)$ can lead to close to maximum distances in $\|r_1 - r_2\|$.

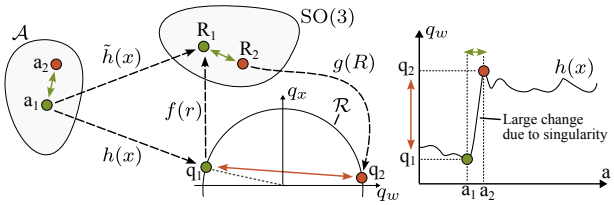

Figure 7. Rotation estimation suffers from double cover: We assume that a small change in the feature space $\mathcal{A}$ corresponds to a small change in $\mathrm{SO}(3)$ (green arrows). For some rotation representations $\mathcal{R}$ such as quaternions, small distances in $SO(3)$ are sometimes mapped to large distances in $\mathcal{R}$ (red arrows) yielding a discontinuous function h (right).

Double Cover A rotation representation doubly covers $\mathrm{SO}(3)$ if every element of $\mathrm{SO}(3)$ is represented by two different elements of $\mathcal{R}$. Examples among the representations are unit-quaternions where $f(q) = f(-q)$, exponential coordinates where $f(\omega) = f((\|\omega\| - 2\pi)\frac{\omega}{\|\omega\|})$, and axis-angle vectors where $[\tilde{\omega}, \phi] = [-\tilde{\omega}, -\phi]$. Double cover leads to a discontinuity in $g(R)$, and as a result, measuring the shortest distance between two representation vectors does not necessarily correspond to the shortest distance in $\mathrm{SO}(3)$. How does this affect gradient-based machine learning?

We can resort to Lipschitz continuity to understand the problems arising from discontinuities. Assume we have two feature vectors $a_1, a_2 \in \mathcal{A}$ that are mapped by $\tilde{h}(x)$ to the nearby elements $R_1, R_2 \in \mathrm{SO}(3)$. The discontinuities in $g(R)$ imply that the distance between two representation vectors $r_1 = g(R_1), r_2 = g(R_2) \in \mathbb{R}^d$ can still be large and $\frac{\|r_2 - r_1\|}{\|R_2 - R_1\|} \xrightarrow[\|R_2 - R_1\| \to 0]{} \infty$. Because of it, there must be points in feature space where $h^*(x) = g \circ \tilde{h}$ has Lipschitz constant $\mathcal{L} = \frac{\|r_2 - r_1\|}{\|a_2 - a_1\|}$ that blows up as $a_2 - a_1$ approaches zero. Naturally, this will cause the loss function's gradient $\nabla_\theta L$ to also blow up. Figure 7 demonstrates this issue for the case of rotation estimation with quaternions yielding discontinuities that almost span across the full width of $\mathcal{R}$.

There seem to be two options: try to fix discontinuities or avoid them in the first place. In the following, we briefly analyze different fixes proposed in the literature, and provide a recommendation for gradient-based learning.

Attempting to fix discontinuities via sin-cos angle coordinates: For any representation that contains angles, we can represent them as $[\cos(\cdot), \sin(\cdot)]$ to avoid the discontinuous jump at the boundaries of the interval $[-\pi, \pi)$. While the above trick is useful for learning with representations of $\mathrm{SO}(2)$ and $\mathrm{SO}(2) \times \mathrm{SO}(2)$ (spherical coordinates), it does not remove discontinuities in $\mathrm{SO}(3)$ representations that arise from double cover. For example, axis-angle still suffer from double cover $f(\omega, \alpha) = f(-\omega, -\alpha)$ even if $\alpha \in \mathrm{SO}(2)$ is replaced by $\cos(\alpha), \sin(\alpha)$.

Attempting to fix discontinuities via distance-picking or computing distances in $\mathrm{SO}(3)$**:** Distance-picking and measuring distances in $\mathrm{SO}(3)$ are common strategies found in literature related to rotation estimation. In distance picking, we choose the minimum distance between elements in $\mathcal{R}$ taking double cover into account e.g. Equations (5) and (6). Alternatively, we can directly compute distances in $\mathrm{SO}(3)$, e.g. via $d_c(f(r_1), f(r_2))$, such that the gradients $\nabla_y d_c(f(y), f(r_2))$ point in directions in $\mathcal{R}$ that increase the metric in $\mathrm{SO}(3)$. However, from the previous discussion and the additional argument provided in Appendix D.4, we conclude the following key insight:

> **Changing the loss does not fix discontinuities:** due to double cover, the target function maps similar features to vastly different rotation representations (one-to-many map). The subsequent issues arising in learning the target function are not fixed using distance picking or computing distances in $\mathrm{SO}(3)$.

This observation is supported by experiments in Alvarez-Tunon et al. (2023) on visual odometry (estimate camera pose from images) and our experiments in Section 5.

Attempting to fix discontinuities via half-space map: Another idea would be to constrain the representation in $\mathcal{R}$ to one *half* such that double-cover is not present (except at the separation hyper-plane). For instance, for quaternions where $f(q) = f(-q)$, we simply flip all vectors with negative scalar dimension as shown in Figure 8. Also, the axis-angle representation and exponential coordinates allow splitting the space in two halves as shown in Figure 3.

Again, the apparent fix has problems for vectors close to the separating hyperplane. Consider the case of quaternions as depicted in Figure 7. Two vectors close to the boundary are far in $\mathcal{R}$ but close in $\mathrm{SO}(3)$, such that the Lipschitz-constant of g can grow unbounded. Empirical evidence in Section 5 and in Saxena et al. (2009) on learning an object's orientation from images supports this. Albeit, for small rotations where $\|\mathbb{I} - f(q)\|_{\mathrm{F}} \leq \sqrt{2}$ with the identity matrix $\mathbb{I}$ amounting to zero rotation as shown in Figure 8 (left), a half-space map is a valid fix as no discontinuities are seen during training (Brégier, 2021). This case is particularly relevant in dynamics simulation, where changes in rotations remain small between timesteps.

Circumventing discontinuities by using 6D or 9D representations: The obvious solution to the above problems is to use high-dimensional representations that have a continuous mapping g (Section 2). While Brégier (2021) provides a detailed analysis of such representations, we limit the discussion to the questions: Why do these representations work well and why does $\mathbb{R}^9$+SVD outperform $\mathbb{R}^6$+GSO?

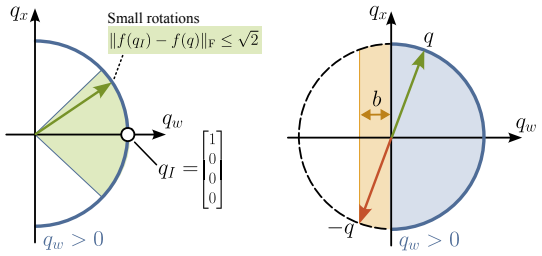

Figure 8. Half-space maps work for "small" rotation estimation (left) and data-augmentation for feature prediction (right). Left: As long as the rotation vectors are within the green cone, there are no discontinuities and the distances behave as expected. Right: To avoid underrepresented input regions, data-augmentation (orange) for vectors close to the half-space boundary can be used.

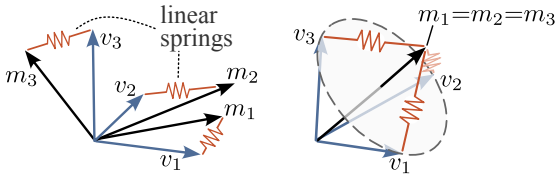

Figure 9. Illustration of $\mathbb{R}^9$+SVD. Left: SVD yields the closest Cartesian frame $R = [v_1, v_2, v_3]$ to the matrix $M = [m_1, m_2, m_3]$. Think of springs that try to pull v_i towards m_i. Right: Singularities arise when $\det(M) = 0$: several R minimize the potential energy.

$\mathbb{R}^9$+SVD is smooth For the $\mathbb{R}^9$+SVD representation of rotation, $f(r) := \mathrm{SVD}^+(M)$ is smooth if $\det(M) \neq 0$ with $r := M$, as it solves the "orthogonal Procrustes problem" using the Frobenius norm (8):

$$\mathrm{SVD}^+(M) := \underset{R \in \mathrm{SO}(3)}{\arg\min} \|R - M\|_{\mathrm{F}}, \tag{12}$$

$$:= \underset{R \in \mathrm{SO}(3)}{\arg\min} \sum_{i=1}^{3} \|v_i - m_i\|^2, \tag{13}$$

where $M = [m_1, m_2, m_3]$ and $R = [v_1, v_2, v_3]$. This method projects M onto the nearest rotation matrix in $\mathrm{SO}(3)$ which we visualize in Figure 9 as spring forces pulling R towards minimum potential energy (13). The potential energy in the springs amounts to the force $\|v_i - m_i\|$ times the distance $\|v_i - m_i\|$. Levinson et al. (2020) analyzed the gradient $\nabla_M L$ in the context of singularities. Their experiments showed that training with data being close to points with $\det(M) = 0$, see Figure 9 right, increases the gradient magnitude, but does not notably impede training.

$\mathbb{R}^6$+GSO and the Procrustes problem As initially pointed out by Brégier (2021), the $\mathbb{R}^6$+GSO representation $r := M = [\nu_1, \nu_2] \in \mathbb{R}^{3\times 2}$ can be seen as a degenerate case of the Procrustes problem such that

$$\mathrm{GSO}(M) := \lim_{\epsilon \to 0^+} \underset{R \in \mathrm{SO}(3)}{\arg\min} \|\mathrm{diag}(1, \epsilon, 0)\, R - M\|_{\mathrm{F}}. \tag{14}$$

Thus, ν_1 has by far the strongest influence on determining R, whereas ν_2 merely determines the rotation around ν_1. In

principle, there are six different ways on how to perform the Gram Schmidt orthonormalization (around the axis e_1-e_2, e_1-e_3, e_2-e_1, e_2-e_3, e_3-e_1, e_3-e_2) which all place importance on different columns of the rotation matrix. $\mathbb{R}^6$+GSO can be reduced to a 5D representation, which in practice seems to perform worse (Zhou et al., 2019).

Comparing $\mathbb{R}^6$+GSO and $\mathbb{R}^9$+SVD Studies by Zhou et al. (2019); Levinson et al. (2020); Brégier (2021), and our experiments confirm that $\mathbb{R}^9$+SVD often outperforms $\mathbb{R}^6$+GSO in rotation estimation. Why is that?

Levinson et al. (2020) attributes better performance of $\mathbb{R}^9$+SVD to the fact that Gaussian noise on M results in twice the error in expectation in R for $\mathbb{R}^6$+GSO compared to $\mathbb{R}^9$+SVD. $\mathbb{R}^6$+GSO predicts the first column of the rotation matrix by normalizing ν_1. In turn noise on ν_1 will directly affect the estimation of R, whereas for $\mathbb{R}^9$+SVD the effect of noise in m_1 also depends on m_2 and m_3. If we interpret the individual representation vectors as the output of different networks, then the SVD layer can be seen as an ensemble model architecture where all three "networks" equally contribute to the prediction whereas $\mathbb{R}^6$+GSO almost fully resorts to the first "network". It is a common strategy to deploy ensemble models rather than a single network as they often show increased *robustness to input noise* and an unlucky initialization of network weights.

Brégier (2021) defined the loss function $L = v_1^T f(r) v_2$ with v_i being vectors pointing in uniform random direction and r denoting either quaternions, $\mathbb{R}^6$+GSO, or $\mathbb{R}^9$+SVD. For random representation vectors $r \sim \mathcal{N}(0,1)^n$, this loss function has a *smaller absolute error to its linear approximation* using $\mathbb{R}^9$+SVD compared to using $\mathbb{R}^6$+GSO or quaternions. Inspired by Brégier (2021), (Dinh et al., 2017), (Gilmer et al., 2022), and Lyle et al. (2023), we take a closer look at the *gradients* of the loss function

$$L(R, r) = \|\text{vec}(R), f(r)\|, \tag{15}$$

with the target rotation $R := \mathbb{I}$ and $f(r)$ denoting either an $\mathbb{R}^6$+GSO or $\mathbb{R}^9$+SVD layer. Assume a network predicts r such that its parameters θ could be updated via the gradient $\nabla_\theta r \circ \nabla_r L$. Then, we are interested in how $\nabla_r L$ varies when using either $\mathbb{R}^6$+GSO or $\mathbb{R}^9$+SVD. To illustrate this, we plot in Figure 10 the path of the representation vectors of $\mathbb{R}^6$+GSO and $\mathbb{R}^9$+SVD when optimizing (15) via gradient-descent with momentum. Due to the way GSO and SVD construct $f(r)$, we observed instabilities in the optimization for $\nu_1 \approx e_3$, $\nu_1 || \nu_2$, $\det(M) = 0$, and $M = -R$.

Moreover, the speed of convergence is notably affected by large ratios between the length of the representation vectors which particularly seems to affect $\mathbb{R}^6$+GSO. We compute the ratios between the length of the gradients of (15) wrt. ν_1, ν_2 or m_1, m_2, m_3. As illustrated in Figure 11 for 20000 uniform randomly initialized $r \in [-2, 2]$, the *gradient ratios remain notably closer to one* for $\mathbb{R}^9$+SVD compared to $\mathbb{R}^6$+GSO. Further, in $\mathbb{R}^9$+SVD, the distributions of gradient ratios are similar. We conclude that in the hypercube with a side-length of four in which we sampled r, when using $\mathbb{R}^6$+GSO compared to $\mathbb{R}^9$+SVD considerably more instances of r exhibit gradients that impede training.

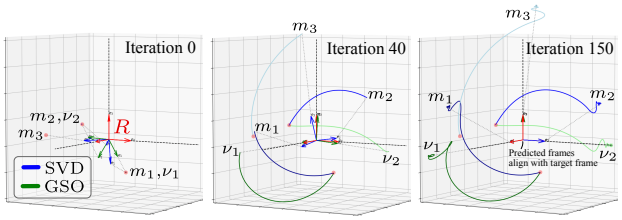

Figure 10. Illustration of how the representation vectors of $\mathbb{R}^6$+GSO $r := [\nu_1, \nu_2]$ and $\mathbb{R}^9$+SVD $r := [m_1, m_2, m_3]$ move through $\mathbb{R}^3$ during optimization. To find an r that minimizes the loss (15) to a target rotation $R := [v_1, v_2, v_3]$, we perform 150 iterations of gradient-descent with momentum and plot the path of the representation vectors. $\mathbb{R}^6$+GSO can only minimize the loss, if ν_1 aligns with v_1 while ν_2 resides on the v_1-v_2-plane.

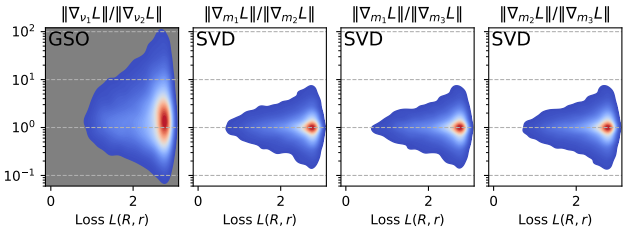

Figure 11. Density plot of the ratios between loss function gradients of $\mathbb{R}^6$+GSO and $\mathbb{R}^9$+SVD. The gradients of the loss (15) are computed for 20000 randomly initialized r.

Another argument why SVD performs better than GSO evolves around the *higher-dimensionality* of the former. In various fields such as position encodings (Dufter et al., 2022), Koopman operator theory (Brunton et al., 2022), and neural ODEs (Zhang et al., 2020), it has been observed that higher dimensional representations may benefit learning. This is particularly the case if the function we discern from data resides in a three-dimensional manifold such as SO(3).

> **For rotation estimation:** use $\mathbb{R}^9$+SVD or $\mathbb{R}^6$+GSO. If the regression targets are only small rotations, using quaternions with a halfspace-map is a good option.

What about directly predicting the entries of R? We find that this is worse than using Procruste-based methods. We hypothesize that this is due to the properties of $\nabla_r L$. When directly predicting rotation matrices with the Chordal distance, the network's predictions must exactly fit the entries of the rotation matrix to reduce the loss. Whereas in $\mathbb{R}^9$+SVD, the *relative arrangement* of the representation vectors to each other determines R.

4.2. What to be aware of for feature prediction?

Feature prediction does not immediately suffer from double cover and a discontinuous g because only the forward function f is part of the learning problem (ii) and f is typically well-behaved. However, there are still problems due to the topological mismatch between $SO(3)$ and low-dimensional representations. First, it is reasonable to employ a halfspace-map to avoid having less samples in each of the halfspaces that would need to be learned separately.

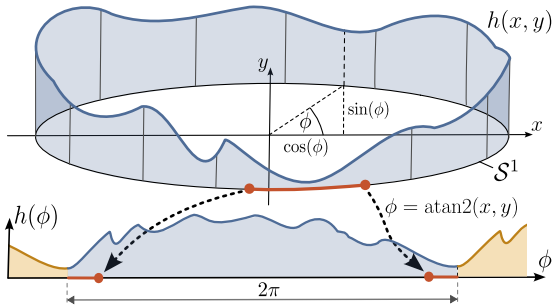

Figure 12. For $\text{SO}(2)$, the function $\phi = f(\cos(\phi), \sin(\phi))$ is continuous, rendering feature prediction feasible. However, some connected sets in $\text{SO}(2)$ are not connected in $\mathcal{R}$ (red). Data augmentation (orange) mitigates the effect of disconnected pre-images.

Fixing low-dim representations using data augmentation: Low-dim representations and the halfspace map introduce a *boundary* into the space (while $SO(N)$ has no boundary). In Figure 12, we illustrate this for $SO(2)$ for clarity. Due to the broken cyclic boundary conditions, we have less samples at the boundary to generalize in these regions. A simple trick is to use data-augmentation to create data beyond the boundary as shown in Figure 8 and 12.

High-dimensional representations: Again, they work out of the box and perform better in practice, as we show in Section 5. More input dimensions allow for a simpler functional form for h. A relation can be drawn to position encodings used in transformers (Dufter et al., 2022). A sin/cos representation (including higher frequencies) works better than using a scalar.

> **For feature prediction:** use $\mathbb{R}^9$+SVD or $\mathbb{R}^6$+GSO. If under memory constraints, quaternions with a halfspace-map and data-augmentation are viable.

5. Experiments

In this section, we empirically assess the earlier discussion and support the recommendations with various experiments on rotation estimation and feature prediction.

Experiment 1 (rotation estimation): Rotation from point clouds The task is to predict from two point clouds $P_1, P_2 \in \mathbb{R}^N$, with $N = 3000$, the rotation R such that

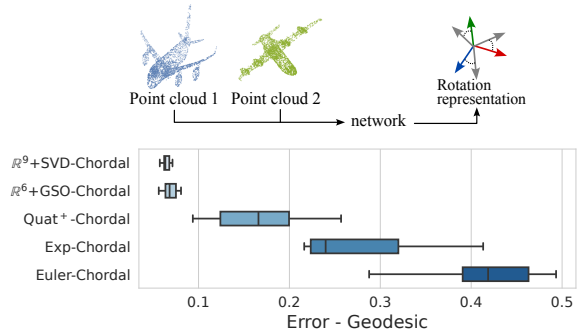

Figure 13. Point cloud alignment experiment. The best test data results from each representation are shown in which $\mathbb{R}^9$+SVD performs best. See Figure 23 for additional results.

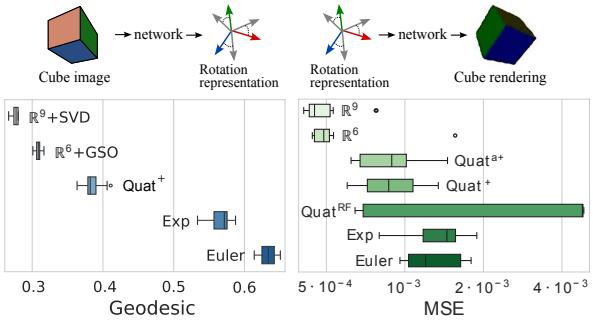

Figure 14. Experiment 2. Left: Rotation estimation from cube image where $\mathbb{R}^9$+SVD performs best. See Figure 24 for additional results. Right: Network trained with pixel-MSE loss renders a cube from a given rotation. While quaternion augmentation slightly reduces the error, $\mathbb{R}^9$+SVD shows superior performance.

$RP_2 = P_1$ (Zhou et al., 2019). We follow the model and dataset generation pipeline as Levinson et al. (2020). Point clouds are extracted from a set of 726 airplane CAD models and undergo uniformly sampled rotations in $SO(3)$ to eliminate any potential rotation bias in the dataset. For testing, 100 pairs are left out. Figure 13 shows that $\mathbb{R}^9$+SVD is best followed by $\mathbb{R}^6$+GSO. Figure 23 shows that for quaternions deploying a half-space map notably improves performance whereas distance picking reduces it.

Experiment 2.1 (rotation estimation): Cube rotation from images In this experiment, we predict the orientation of a colorful cube represented as a rendered image. The experimental setup is explained in Appendix E.2. The results in Figure 14 (left) paint the the same picture as before showing that $\mathbb{R}^9$+SVD and $\mathbb{R}^6$+GSO perform notably better than low-dimensional representations.

Experiment 2.2 (feature prediction): Cube rotation to images Now, we look at the inverse problem to Experiment 2.1 predicting the cube's image from an representation of its orientation. The experiment is further described in Appendix E.2. The results in Figure 14 (right) show that $\mathbb{R}^9$+SVD performed notably better than $\mathbb{R}^6$+GSO while quaternions benefit from deploying a half-space map.

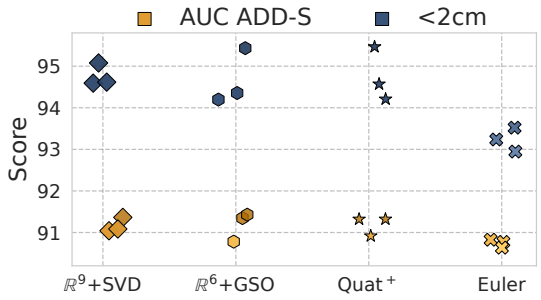

Figure 15. 6D-pose estimation from images on YCB-Video dataset. The metrics are averaged across the 21 objects. Euler angle perform worst while the other representations perform similarly.

Experiment 3 (rotation estimation): 6D object pose estimation from RGB-D images In this experiment, we re-train the network of Wang et al. (2019) using different rotation representations. Simply put, the network predicts a translation and rotation representation which is used to transform an object's point cloud such that it aligns with the ground truth point cloud. This model has become a common baseline for object pose estimation from a single RGB-D camera image. Figure 15 shows the average accuracy across all objects for three seeds. Euler angles perform notably worse than $\mathbb{R}^9$+SVD and $\mathbb{R}^6$+GSO whereas quaternions perform similarly. We hypothesise, that quaternions perform well due to the data set mostly containing small angles. For details, see Appendix E.3.

Experiment 4 (feature prediction): $\mathrm{SO}(3)$ as input to Fourier series In this experiment, we analyze how function complexity affects feature prediction. The target function is chosen as

$$h^*(x) = \sum_{i=1}^{n_b} \left(A_k \cos\left(\frac{k\pi t(R)}{L}\right) + B_k \sin\left(\frac{k\pi t(R)}{L}\right)\right) \tag{16}$$

with period $L = 2$, fourier parameters A_k, B_k, and $t : \mathrm{SO}(3) \to \mathbb{R}$ being a 2-layer MLP with ReLu activations. Both the parameters of $t(\mathrm{vec}(R))$ and the Fourier parameters are randomly initialized. If we increase n_b, then the complexity of (16) increases in expectation. As further detailed in Sec. E.4, we approximate each randomly generated target function with an MLP which has a rotation representation as input. Figure 16 shows the training results from $n_b = 1$ up to $n_b = 5$ for 100 target functions each. We keep the number of points fixed to 800, 200, and 1000 for the train, the validation, and the test data set, respectively. As we increase n_b, the target function becomes on average more wiggly and the model MSEs increase. Augmentation of quaternions (augmentation threshold b=0.1) has a notable effect for $n_b = 1$ to $n_b = 3$. Albeit, for larger n_b, the error arising from having no data points beyond the domain boundaries seem to be of minor importance.

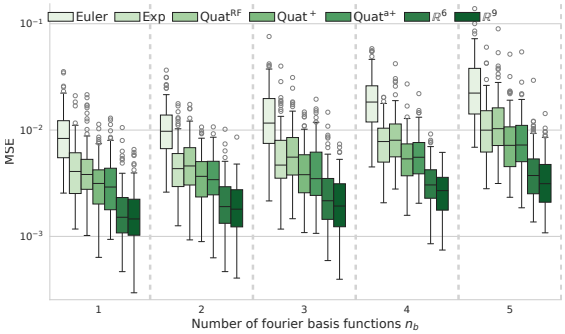

Figure 16. Feature prediction of a Fourier series that has 3D rotations as input. For increasing n_b, the distribution over test errors of 100 randomly sampled target functions is shown. Augmenting quaternions (Quat^{a+}) notably improves prediction accuracy.

6. Conclusion

In this work, we examine recent works on learning with rotations and consider the case of rotation representations in the input (feature prediction) or in the output (rotation estimation) of a regression task. We find it useful to consider the Lipschitz continuity of the function g mapping from SO(N) to the representation space $\mathcal{R}$. We show how this mapping enters the learning problem and see why rotation representations with three or four parameters impede learning by inevitably introducing discontinuities into the target function, if the rotation is in the model's output. We show that these discontinuities cannot be resolved by distance-picking, measuring distances in $\mathrm{SO}(3)$, or half-space maps. In contrast, $\mathbb{R}^9$+SVD and $\mathbb{R}^6$+GSO stand out as superior choices when learning with rotations. In cases involving small angles, unit-quaternions properly mapped to a half space can be practical for rotation estimation.

When rotations appear in the inputs of the regression task, discontinuities do not hinder learning. Nevertheless, high-dimensional representations are superior in practice. When memory is constrained such that low-dimensional representations are preferred, the double cover property needs to be taken care of using half-space maps. Further, data augmentation reduces the impact of non-cyclic boundaries, such that viable performance improvements can be achieved.

Numerous extensions to the discussed topics are being proposed. Notably, Peretroukhin et al. (2020) represents rotations through a symmetric matrix that defines an antipodally symmetric distribution over quaternions effectively mitigating the problems arising due to double cover. This approach outperformed $\mathbb{R}^6$+GSO on various benchmarks. Chen et al. (2022) notably improved prediction performance of $\mathbb{R}^6$+GSO, $\mathbb{R}^9$+SVD, and (Peretroukhin et al., 2020) layers whose predictions are fed into an L2 loss by adjusting the gradients during the backward pass which marginally increases computation times.

Acknowledgments

The authors extend their gratitude to Shukrullo Nazirjonov whose curiosity on learning with rotations served as inspiration for this work.

This work was supported by the ERC - 101045454 REAL-RL and the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ: 01IS18039B). Georg Martius is a member of the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Alvarez-Tunon, O., Brodskiy, Y., and Kayacan, E. Loss it right: Euclidean and riemannian metrics in learning-based visual odometry. In *ISR Europe; 56th International Symposium on Robotics*, pp. 107–111, 2023.

Bosch, M. T., 2020a. URL https://marctenbosch.com/quaternions/code.htm.

Bosch, M. T. Let's remove quaternions from every 3d engine, 2020b. URL https://marctenbosch.com/quaternions/.

Brégier, R. Deep regression on manifolds: a 3d rotation case study. In *2021 International Conference on 3D Vision (3DV)*, pp. 166–174. IEEE, 2021.

Brunton, S. L., Budišić, M., Kaiser, E., and Kutz, J. N. Modern koopman theory for dynamical systems. *SIAM Review*, 64(2):229–340, 2022.

Chen, J., Yin, Y., Birdal, T., Chen, B., Guibas, L. J., and Wang, H. Projective manifold gradient layer for deep rotation regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6646–6655, 2022.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pp. 1019–1028. PMLR, 2017.

Dufter, P., Schmitt, M., and Schütze, H. Position information in transformers: an overview. *Computational Linguistics*, 48(3):733–763, 09 2022.

Euler, L. Du mouvement de rotation des corps solides autour d'un axe variable. *Mémoires de l'académie des sciences de Berlin*, pp. 154–193, 1765.

Gilmer, J., Ghorbani, B., Garg, A., Kudugunta, S., Neyshabur, B., Cardoze, D., Dahl, G. E., Nado, Z., and Firat, O. A loss curvature perspective on training instabilities of deep learning models. In *International Conference on Learning Representations*, 2022.

Grassia, F. S. Practical parameterization of rotations using the exponential map. *Journal of graphics tools*, 3(3): 29–48, 1998.

Hartley, R., Trumpf, J., Dai, Y., and Li, H. Rotation averaging. *International journal of computer vision*, 103: 267–305, 2013.

Huynh, D. Q. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35:155–164, 2009.

Levinson, J., Esteves, C., Chen, K., Snavely, N., Kanazawa, A., Rostamizadeh, A., and Makadia, A. An analysis of svd for deep rotation estimation. *Advances in Neural Information Processing Systems*, 33:22554–22565, 2020.

Llanas, B., Lantarón, S., and Sáinz, F. J. Constructive approximation of discontinuous functions by neural networks. *Neural Processing Letters*, 27:209–226, 2008.

LLC., M. Cmu graphics lab motion capture database, 2024. URL http://mocap.cs.cmu.edu/.

Lyle, C., Zheng, Z., Nikishin, E., Pires, B. A., Pascanu, R., and Dabney, W. Understanding plasticity in neural networks. In *International Conference on Machine Learning*, pp. 23190–23211. PMLR, 2023.

Macdonald, A. *Linear and geometric algebra*. Alan Macdonald, 2010.

Mäkinen, J. Rotation manifold so(3) and its tangential vectors. *Computational Mechanics*, 42:907–919, 2008.

Pepe, A., Lasenby, J., and Chacón, P. Learning rotations. *Mathematical Methods in the Applied Sciences*, 2022.

Peretroukhin, V., Giamou, M., Rosen, D. M., Greene, W. N., Roy, N., and Kelly, J. A smooth representation of SO(3) for deep rotation learning with uncertainty. In *Proceedings of Robotics: Science and Systems*, Jul. 12–16 2020.

Saxena, A., Driemeyer, J., and Ng, A. Y. Learning 3d object orientation from images. In *2009 IEEE International conference on robotics and automation*, pp. 794–800. IEEE, 2009.

Stuelpnagel, J. On the parametrization of the three-dimensional rotation group. *SIAM review*, 6(4):422–430, 1964.

Terzakis, G., Lourakis, M., and Ait-Boudaoud, D. Modified rodrigues parameters: an efficient representation of orientation in 3d vision and graphics. *Journal of Mathematical Imaging and Vision*, 60:422–442, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17:261–272, 2020.

Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., and Savarese, S. Densefusion: 6d object pose estimation by iterative dense fusion. 2019.

Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *Robotics: Science and Systems (RSS)*, 2018.

Xu, Z. and Cao, F. The essential order of approximation for neural networks. *Science in China Series F: Information Sciences*, 47:97–112, 2004.

Xu, Z.-B. and Cao, F.-L. Simultaneous lp-approximation order for neural networks. *Neural Networks*, 18(7):914–923, 2005.

Zhang, H., Gao, X., Unterman, J., and Arodz, T. Approximation capabilities of neural odes and invertible residual networks. In *International Conference on Machine Learning*, pp. 11086–11095. PMLR, 2020.

Zhou, Y., Barnes, C., Lu, J., Yang, J., and Li, H. 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. Notation

Vectors are denoted as $a = [a_1, a_2, a_3] \in \mathbb{R}^{3\times 1}$. The identity matrix of size three is denoted as $\mathbb{I}$. We denote the n dimensional unit sphere as $\mathcal{S}^n = \{x \in \mathbb{R}^{n+1} : \|x\| = 1\}$. Given a vector $a \in \mathbb{R}^d$, $\text{diag}(a) \in \mathbb{R}^{d\times d}$ denotes a diagonal matrix with the entries of a on its diagonal. Given a vector $a = [a_1, a_2, a_3] \in \mathbb{R}^3$, $a \times a = [a]_\times \cdot a$ where

$$[a]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{17}$$

Given a matrix $M \in \mathbb{R}^{m\times n}$, the operation $\text{vec}(M)$ rearranges the elements of M as a vector such that $\text{vec}(M) \in \mathbb{R}^{mn\times 1}$. Given a scalar $\phi \in \mathbb{R}$, its absolute value is denoted as $|\phi| \geq 0$.

B. Matrix representation of SO(3)

$\text{SO}(3)$ denotes the *special orthogonal group*. This name comes from the fact that all rotation matrices are special ($det(M) = 1$) and orthogonal. In three dimensions, every rotation is described by a rotation matrix $M \in \text{SO}(3)$ being a real-valued matrix of the form

$$M = \begin{bmatrix} | & | & | \\ m_1 & m_2 & m_3 \\ | & | & | \end{bmatrix} \in \mathbb{R}^{3\times 3} \tag{18}$$

whose three column-vectors m_1, m_2, m_3 span a Cartesian frame such that

$$\|m_i\| = 1 \text{ for } i = 1, 2, 3, \;\; m_3 = m_1 \times m_2. \tag{19}$$

As illustrated in Fig. 17, rotations are commonly represented by a vector $a \in \mathbb{R}^3$ transformed by matrix M to produce $b = Ma \in \mathbb{R}^3$. Alternatively, we gain geometric intuition on the rotation matrix itself if we imagine the vector a to be described wrt. to a frame $\{A\}$, writing ${}^Aa = [a_1, a_2, a_3]$, and the vector ${}^Ba$ being obtained as

$${}^Ba = M\,{}^Aa = m_1a_1 + m_2a_2 + m_3a_3 \tag{20}$$

where $\{B\}$ denotes a frame rotated relative to $\{A\}$. In turn, we can think of the rotation matrix in terms of its columns $m_1 := {}^{\text{B}}e_1^{\text{A}}, m_2 := {}^{\text{B}}e_2^{\text{A}}, m_3 := {}^{\text{B}}e_3^{\text{A}}$ being the unit vectors e_i of $\{A\}$ expressed relative to a rotated frame $\{B\}$. Simply put, the columns of a rotation matrix span a Cartesian frame relative to another Cartesian frame.

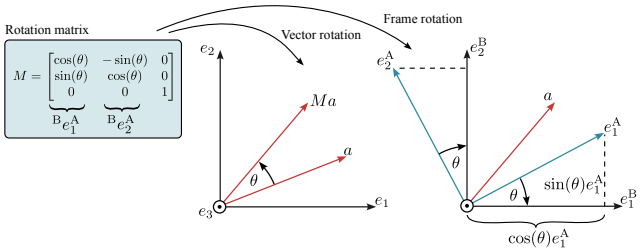

Figure 17. A rotation matrix $M \in SO(3)$ rotates a vector $a \in \mathbb{R}^3$ to $Ma \in \mathbb{R}^3$. Alternatively, the multiplication of a frame's unit vectors with M shows that the columns of the rotation matrix denote the frame's unit vectors after rotation.

C. Intuition on some terminology from linear algebra

"Orthogonal" refers to $MM^T = M^TM = I$ such that the columns of the matrix are orthogonal to each other and have unit length.

The **dot product** between two vectors $a \cdot b = \|a\|\|b\|\cos(\theta)$ (θ is the angle between a and b) is zero if the vectors are orthogonal, one if they are colinear unit vectors pointing in the same direction, and -1 if they are colinear unit vectors pointing in opposite directions. If a and b are unit vectors, then the inner product corresponds to $\cos(\theta)$.

The **determinant** of a 3D matrix measures the signed volume of the parallelepiped spanned by the column vectors of the matrix.

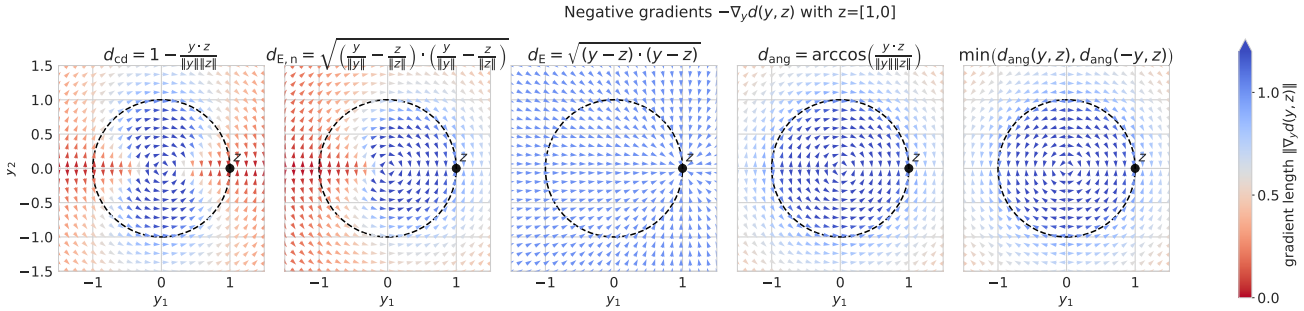

Figure 18. Given a fixed target vector $z = [1, 0] \in \mathcal{S}^1$, the negative gradients $-\nabla_y d(y, z)$ are plotted for different $y = [y_1, y_2] \in \mathbb{R}^2$. For the Euclidean distance with input normalization and cosine distance, the gradient $\nabla_y d(y, z)$ is zero when y is antipodal to z. For the Euclidean distance the gradients point from z to y. For the angular distance, the gradients are tangential to the unit-circle. When training a model $y = h(a, \theta)$, we use $\nabla_y d(y, z)$ to adjust θ such that y is close to the target vector z.

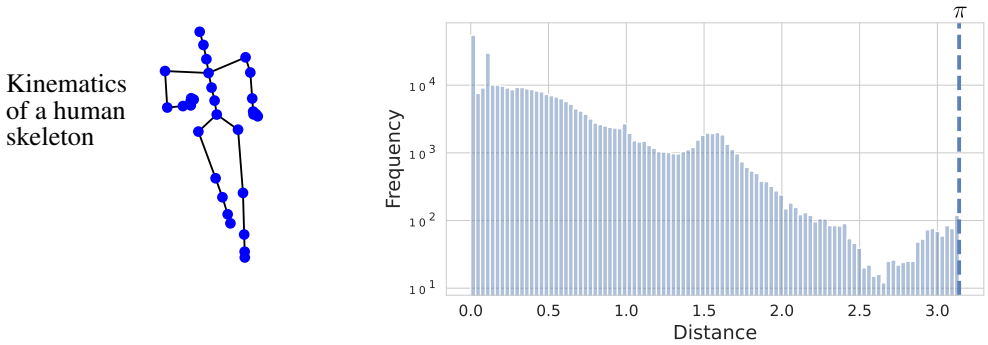

Figure 19. **Left:** Illustration of a human skeleton as contained in the CMU MoCap database (LLC., 2024). The database contains marker position of human skeletons and the corresponding rotations between the lines. **Right:** Distribution of the distances between the identity rotation and the rotations in the CMU MoCap database (LLC., 2024). This data base is used in the rotation estimation experiments of Zhou et al. (2019); Levinson et al. (2020); Brégier (2021); Pepe et al. (2022). In $\mathrm{SO}(3)$, two rotations are maximally π away from each other when using the Geodesic distance. In this data set, most rotations are close to the identity rotation such that the singularities inherent in low-dimensional rotation representations may not significantly affect rotation estimation.

"Special" refers to $\det(M) = 1$. If $\det(M) = 1$, then the column vectors of M form a right-hand coordinate system. Orthogonal matrices with determinant of one are called **proper rotations**, whereas orthogonal matrices with determinant of negative one are called **improper rotations**. Improper rotations combine rotation with reflection, the product between two improper rotations is a proper rotation. We encounter improper rotations in the description of rotations via geometric algebra where rotations are described by means of two reflections.

D. Additional aspects on learning with rotations

D.1. Why learn with rotation representations that have three or four parameters?

Computational efficiency A representation with 3 instead of 9 parameters may significantly reduce memory consumption. When advocating for low-dimensional rotation representations, it is tempting to resort to the "curse of dimensionality" which relates to the volume of the space between data points growing exponentially with increasing data dimensionality. However, $\mathrm{SO}(3)$ rotations reside on a three-dimensional manifold in $\mathbb{R}^9$ such that it is unclear if the curse of dimensionality aggravates learning for higher-dimensional rotation representations.

Forward dynamics If a network predicts the forward dynamics of a multi-body system, small errors in the prediction of the body's rotation over time may accumulate. In turn, the predicted state eventually does not remain in the proximity of the three-dimensional manifold in $\mathbb{R}^9$ on which the training data has been collected. Mäkinen (2008) provides comprehensive theoretical analysis on finite changes in rotations.

D.2. How do we augment quaternions for feature prediction?

The following code performs data augmentation of a data set containing unit-quaternions to improve learning with rotations in the model's output:

```
x_quat = x_rot.as_quat(canoncial=True)
epsilon = 0.1
condition = x_quat[:, -1] < epsilon
x_quat = np.r_[-1 * x_quat[condition,:], x_quat]
y_features = np.r_[y_features[condition,:], y_features]
```

Assume that each row of the array `x_quat`$\in \mathbb{R}^{m\times 4}$ denotes a unit-quaternion. Here, we follow the convention in SciPy that the last dimension of the quaternion is its scalar part. To every quaternion a feature is being associated which is stored in the array `y_features`$\in \mathbb{R}^{m\times n}$. As shown below, we first ensure that all quaternions lie on the half unit sphere such that $q_w \geq 0$ e.g. by using ScipPy's rotation library `x_rot.as_quat(canoncial=True)`. Then we check which quaternions have a scalar part $q_w <$`epsilon` and store the indices of these quaternions in an array. These quaternions are then mapped to the other half-sphere by multiplication with -1 and concatenated to the data set. `y_features` is also augmented using the same indices.

Batch-wise augmentation of quaternions In the experiment implementation, instead of following the above approach for augmenting quaternions, we augment solely the rotations in the current batch via the PyTorch code:

```
x_quat[torch.logical_and(torch.rand(x_quat.size(0)) < 0.5,
                         x_quat[:, 3] < 0.1)] *= -1
```

Before running the above augmentation, we ensured that `x_quat` is a torch array of size $m \times 4$ that only contains canonical quaternions where m denotes the batch size. As this approach only operates on the current batch it is more memory-efficient but computationally more costly. Further, with batch-wise augmentation a quaternion q and its complement $-q$ cannot simultaneously occur in the augmented batch if q appears only once in the data.

D.3. Are rotors underrated?

Pepe et al. (2022) contends that using bi-vectors (3D) and rotors (4D) for rotation estimation might rival rotation representations employing the Gram-Schmidt orthonormalization (6D). However, bi-vectors / rotors are in fact computationally equivalent to exponential coordinates / quaternions as shown in a detailed code comparison by Bosch (2020a). While a geometric algebra perspective on rotations (including rotors and bi-vectors) may be more intuitive than using quaternions, as exemplified by Macdonald (2010, p. 87-91) and Bosch (2020b), we remain cautious about expecting significant performance differences for rotation estimation.

Additionally, while Table 2 in (Pepe et al., 2022) suggests that networks can learn the map from rotors to bi-vectors, these representations are not exempt from the singularities in $g(R)$ causing $h(x)$ to become discontinuous during rotation estimation. Further, Pepe et al. (2022) observed that rotors obtained using the Caley-transform performed differently than rotors obtained using the matrix-exponential. We hypothesize that one of these maps performs a half-space map while the other does not.

We advocate for further analysis that considers the proximity of representation vectors to singularities, measures all errors in $\mathrm{SO}(3)$ rather than in rotor space, and tries using a half-space map for quaternions and exponential coordinates. One experiment found in Zhou et al. (2019); Levinson et al. (2020); Brégier (2021); Pepe et al. (2022) is "Inverse Kinematics with CMU MoCap data" (see Sec. E.5), for which we analysed how far the rotations are from the unit rotation. We hypothesized that for some joints the anatomy of human kinematics does not support large angle ranges and in turn the data set contains mostly small rotations (recall that a rotation $R \in \mathrm{SO}(3)$ is small if $\|\mathbb{I} - R\|_{\mathrm{F}} \leq \sqrt{2}$). Indeed, as shown in Figure 19, this data set contains significantly more small rotations that are close to the unit rotation which reduce the effect of singularities on learning with rotation representations with three or four dimensions.

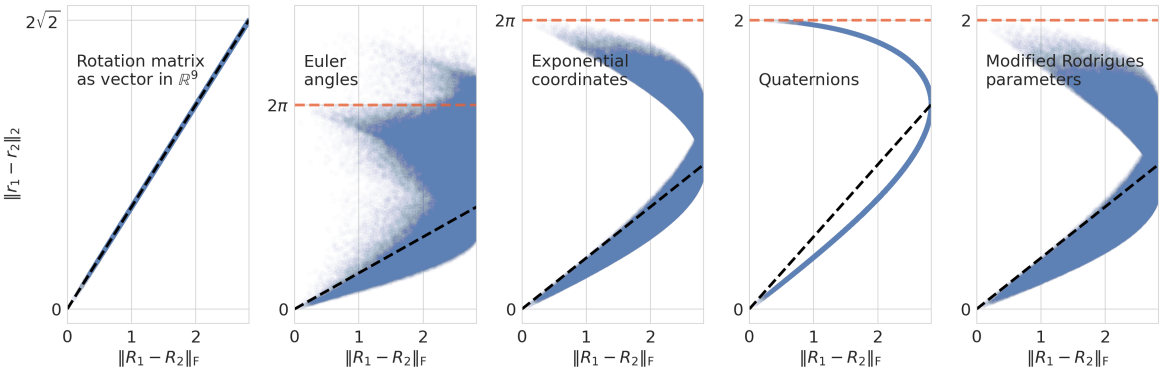

Figure 20. Chordal distances $\|R_1 - R_2\|_F$ between randomly sampled $R_1, R_2 \in \mathrm{SO}(3)$ and distance $\|r_1 - r_2\|$ between corresponding rotation representations $r_1 = g(R_1), r_2 = g(R_2)$. The full width of $\mathcal{R}$ is marked by red lines. Ideally, the ratios between the distances, aka the Lipschitz constant of $g(R)$, is close to the black lines whose slope amounts to the ratio between the minimum width of $\mathcal{R}$ and $\mathrm{SO}(3)$. For Euler angles how much a distance in $\mathrm{SO}(3)$ is altered to a distance in $\mathcal{R}$ strongly depends on where R_1, R_2 lie in $\mathrm{SO}(3)$. In comparison, for unit-quaternions, distances in $\mathrm{SO}(3)$ of a certain magnitude of any R_1, R_2 correspond to similar distances in $\mathcal{R}$.

D.4. How does distance picking and computing distances in $\mathrm{SO}(3)$ affect rotation estimation?

The term distance picking relates to using metrics such as (5) and (6) for rotation estimation. As pointed out by (Huynh, 2009), these functions are pseudo-metrics in $\mathcal{R}$ but metrics in $\mathrm{SO}(3)$. How do the prediction of a neural network $h(x, \theta)$ look like that is trained using such a metric? At the beginning of training the network's parameter are randomly initialized and the output of the network points in different direction of $\mathcal{R}$. Then during training the networks parameters are being updated to move the network's output close to the rotation representation or their negative complement that is closest according to e.g. (5). Now for simplicity's sake, assume that the training data consists of two features a_1, a_2 which reside close to each other such that $a_1 = a_2 + \epsilon$. Yet, due to an unlucky initialization $q \approx h(a_1, \theta)$ and $-q \approx h(a_2, \theta)$. In this case, the optimizer deploying (5) adjusts the network's parameters to move $\hat{q}_1 = h(a_1, \theta)$ close to q and $\hat{q}_2 = h(a_2, \theta)$ close to $-q$. In turn, the network's ability to interpolate between features that map to similar rotations is severely hindered while the network's gradients may vary significantly in magnitude.

D.5. Additional aspects on learning with rotations

Besides the continuity of $g(a)$, numerous other aspects affect learning with rotations. In particular, the distribution of the training data and observability of the object should be analyzed prior to training.

If only a small part of the rotation space is present within the training dataset, one cannot generalize to the full rotation space. This problem has been commonly experienced by researchers training models on the YCB-Video dataset where most objects are biased toward standing upright. Similarly, the CMU MoCap database (LLC., 2024) contains mostly small angle ranges.

The second aspect is, the observability of the task at hand. If objects are (partially)-occluded or symmetrical, the learning problem may be ill-posed as pointed out by (Saxena et al., 2009).

D.6. How does the ratio between distances induced by $g(r)$ look like for axis-angles and exponential coordinates?

Figure 20 shows also the ratio between the Chordal distance $\|R_1 - R_2\|_F$ of randomly sampled rotation matrices $R_1, R_2 \in \mathrm{SO}(3)$ relative to the corresponding distance in $\mathcal{R}$ both for exponential coordinates and axis-angles. For Euler angles the ratio between the distances strongly depends on where the rotation matrices are in $\mathrm{SO}(3)$. For exponential coordinates and modified Rodrigues parameters this distance ratio looks similar to unit-quaternions yet is clearly not as neatly aligned. Modified Rodrigues parameters $[\tilde{\omega}, \bar{\alpha}]$ are obtained by taking the angle α in the axis angle representation and transforming it to $\bar{\alpha} = \tan(\alpha/4)$. Modified Rodrigues parameters are bijectively mapped to quaternions via stereographic projection as succinctly illustrated in (Terzakis et al., 2018).

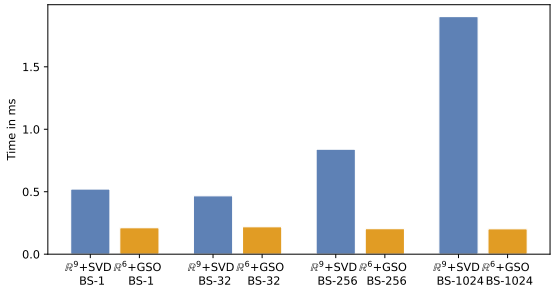

Figure 21. Experiment 2.2: Timing comparison of $\mathbb{R}^6$+GSO and $\mathbb{R}^9$+SVD to $\mathrm{SO}(3)$ mapping for different batch sizes $[1, 32, 256, 1024]$ on a Nvidia RTX3060.

D.7. How does $\mathbb{R}^9$+SVD and $\mathbb{R}^6$+GSO compare in terms of computation times?

We previously, concluded that for most applications, one should use the continuous rotation representation. In the following, we analyze the additional computational cost required for training and inference when using the $\mathbb{R}^6 + GSO$ and $\mathbb{R}^9 + SVD$ rotation representation. We timed the forward pass for 100 samples of the GSO and SVD mapping operations within Experiment 2.1. We followed the best practices and took into account the asynchronies execution on the GPU, and warm-started the accelerator by inferencing 100 samples before timing the mapping functions. All numbers are reported using an Nvidia RTX3060.

The results can be seen in Figure 21. The SVD is computationally more expensive than the GSO. The GPU-based $\mathbb{R}^9 + SVD$ takes around $0.5\,\mathrm{ms}$ for a batch size of 1. However, there exist multiple CPU-optimized SVD implementations that can decrease the inference time for small batch size ($0.002\,\mathrm{ms}$ `https://github.com/ericjang/svd3`), or one can resign to using Jax for better efficiency. When looking at a batch size of 1024, the SVD only takes $2\,\mathrm{ms}$. Specifically crucial for deploying neural networks is the increase in latency for small batch sizes (e.g. running an object pose estimator on a robot), which is nearly neglectable when using a CPU-based SVD implementation. Another critical aspect is the reduced throughput during training which, when using high-capacity neural network architectures like CNNs, Transformers, or Diffusion Models, is neglectable as well (e.g. $200\,\mathrm{ms}$ ResNet152 vs $2\,\mathrm{ms}$ SVD). However, specifically when training very small networks (e.g. 2-layer MLP), it can be beneficial to directly optimize on $\mathbb{R}^9$ (without SVD) and without GSO leading to only a minor performance decrease (see Figure 24).

In summary,we conclude that for most applications, the overhead in terms of compute when using $\mathbb{R}^9 + SVD$ compared to $\mathbb{R}^6 + GSO$ is small. We highlight the clear trade-off between performance, increased latency, and throughput that one has to take into consideration.

E. Further experiment details

The project code is available at: github.com/martius-lab/hitchhiking-rotations

E.1. Details of Experiment 1 (Rotation estimation): Rotation from point clouds

The dataset creation consists of first sampling 3000 points from the 726 airplane CAD models provided by the ModelNet dataset and secondly rotating them with rotations uniformly sampled from $SO(3)$, using "SciPy" (Virtanen et al., 2020). The 726 pairs of rotated point clouds and their rotations are split into 626 training and 100 testing samples. To get the different rotation representations, the rotations are transformed into Euler angles (xyz-intrinsic rotations), canonical quaternions (mapped to have positive scalar), and exponential coordinates. For $\mathbb{R}^9$+SVD and $\mathbb{R}^6$+GSO the matrix form is used, and the representation is captured by a mapping function as explained below.

The architecture of the model used for learning follows exactly the architecture described in Levinson et al. (2020) and Zhou et al. (2019). First, point clouds are embedded with a simplified PointNet (4 MLP layers 64, 128, 256, 1024) followed by a global max-pooling. This is then regressed via two LeakyReLu-activated MLP(512) layers with dropout followed by an MLP(d) which outputs the d-dimensional vector of the corresponding representation.

Figure 22. Experiment 2.2: Cube Rendering Examples.

During training, we minimize either the mean absolute error (MAE) or mean squared error (MSE). For quaternions, we also experiment with distance picking (-dp) as training loss. The $\mathbb{R}^9$+SVD and GSO representation are trained by projecting the predicted vector $\mathbb{R}^d$ to $\mathbb{R}^9$ and minimizing the chosen loss there. We incorporate early stopping in our training process with a patience of 10 epochs and a maximum of 100 epochs to ensure a fair and effective training of diverse representations.

E.2. Details of Experiment 2: Cube image to / from rotation

We create a dataset of differently rotated cubes at the same positions with respect to a fixed camera. The dataset is generated once and kept fixed across training seeds. The training, validation, and test dataset consists of 2048 image/rotation pairs. The rotations, used to render the images are sampled uniformly distributed for both datasets. The images are rendered using the MuJoCo simulator (Todorov et al., 2012) at a resolution of 64×64. In all experiments, we use the Adam optimizer and train for up to 1000 epochs. We early stop the training if no improvement is achieved over 10 validation epochs. We report all metrics across 10 random seeds, for the best-performing validation model on the test dataset. All hyperparameters are constant across the same experiment. For the orientation prediction task, we use a learning rate of 0.001 and a batch size of 32. The network is given by a 3-layer MLP, with input size $64 * 64 * 3$, hidden dimension $[256, 256]$, ReLU activation functions, and the output size is determined by the dimension of the rotation representation.

Within the main paper (Experiment 2.1, Figure 14), we use the geodesic distance as a training objective and report the Chordal distance. In Figure 24, we report the geodesic distance as well as the Chrodal distance and compare different rotation representations and training objectives.

When training on the Chordal or geodesic distance, we at first map the predicted representation to $\mathbb{R}^9$. The MAE and MSE are directly computed between the predicted rotation representation and the target rotation, without mapping to $\mathbb{R}^9$. All metrics for $\mathbb{R}^9$ can be optimized with and without SVD. For $\mathbb{R}^6$, only the MAE and MSE loss can be optimized without applying GSO. For quaternions, we additionally train on the cosine distance (CD) and use MSE distance picking (MSE-DP).

Generally, we found that the continuous representations strongly outperform quaternion, exponential coordinate, and Euler angle representations. One of the key findings is, that by applying an output mapping to the rotation representation and using a metric defined on $\mathbb{R}^9$ (e.g, the Chordal or geodesic distance), one cannot overcome the limitations intrinsic to the output representation of the network.

For the quaternion representation, we observe that first mapping to $\mathbb{R}^9$ and using the geodesic distance and the Chordal distance leads to an overall better performance compared to cosine distance (CD) and applying distance picking on the MSE (MSE-DP). One would hypothesize when evaluating the Chordal distance, one should optimize for the Chordal distance during training. However, for $\mathbb{R}^9$, $\mathbb{R}^6$ optimizing for the geodesic distance leads to better performance. We would recommend the reader experimentally try out both objective functions.

For the feature prediction/image generation task, we use a learning rate of 0.01 and a batch size of 128. The network is given by a Convolutional Neural Network (CNN), consisting of one LinearLayer and 4 blocks of ConvTranspose2d, BatchNorm2d and ReLU activation function. The loss function and evaluation metric is given by the pixel-wise MSE.

E.3. Details of Experiment 3 (Rotation estimation): 6D object pose estimation from RGB-D images

We use the implementation provided of Wang et al. (2019) and test the performance of the network on the YCB-Video Dataset consisting of 21 different objects and follow the evaluation of Xiang et al. (2018), reporting the Area under the Curve of the ADD-S (AUC) and the percentage of objects with an ADD-S smaller than $2\,\mathrm{cm}$ (<$2\,\mathrm{cm}$). We only consider the initial pose estimation without refinement. Therefore, we adapt the *per-pixel* method while solely altering the network output between different rotation representations. We use a total of 3 seeds because of the high training cost. Table 2 reports the mean and standard deviation.

Table 2. Experiment 3: Per-object 6D pose estimation results, measured by Area under the Curve of the ADD-S (AUC) and percentage of predictions with a ADD-S lower the 2 cm (<2 cm) on the YCB-Video dataset. The averages are provided in the last row. The asterisk '*' indicates objects with symmetry.

	Euler		**$\mathbb{R}^6$+GSO**		**$\mathbb{R}^9$+SVD**		**Quat per-pixel**	
	<2 cm	AUC	<2 cm	AUC	<2 cm	AUC	<2 cm	AUC
Master chef can	100.0 ±0.0	95.0 ±0.2	100.0 ±0.0	94.9 ±0.39	100.0 ±0.0	95.1 ±0.04	100.0 ±0.0	95.3 ±0.14
Cracker box	99.2 ±0.46	92.1 ±0.41	98.2 ±1.75	93.3 ±0.2	99.0 ±0.29	92.4 ±0.12	99.4 ±0.3	92.8 ±0.31
Sugar box	99.8 ±0.16	95.2 ±0.09	100.0 ±0.0	95.4 ±0.15	100.0 ±0.04	95.4 ±0.26	100.0 ±0.0	95.4 ±0.28
Tomato soup can	96.9 ±0.0	93.6 ±0.23	96.9 ±0.0	93.8 ±0.13	96.9 ±0.0	93.7 ±0.04	96.9 ±0.0	93.9 ±0.12
Mustard bottle	100.0 ±0.0	95.6 ±0.71	99.8 ±0.26	95.6 ±0.52	100.0 ±0.0	95.8 ±0.17	100.0 ±0.0	96.0 ±0.2
Tuna can	100.0 ±0.0	95.4 ±0.45	100.0 ±0.0	95.7 ±0.08	100.0 ±0.0	95.9 ±0.26	100.0 ±0.0	95.2 ±0.41
Pudding box	100.0 ±0.0	94.1 ±0.06	100.0 ±0.0	93.7 ±0.32	99.8 ±0.22	93.7 ±0.68	99.8 ±0.22	94.1 ±0.55
Gelatin box	100.0 ±0.0	96.7 ±0.26	100.0 ±0.0	97.2 ±0.33	100.0 ±0.0	97.0 ±0.25	100.0 ±0.0	97.0 ±0.2
Potted meat can	92.5 ±0.12	89.7 ±0.09	92.9 ±0.12	89.6 ±0.46	93.0 ±0.16	89.8 ±0.11	92.8 ±0.18	89.9 ±0.47
Banana	88.2 ±7.13	90.0 ±1.39	93.2 ±2.2	92.2 ±0.82	93.9 ±1.63	92.3 ±0.41	91.5 ±2.48	91.2 ±0.91
Pitcher	100.0 ±0.0	93.3 ±0.2	99.4 ±0.83	93.3 ±0.46	99.9 ±0.08	93.5 ±0.53	100.0 ±0.0	93.6 ±0.57
Bleach	98.4 ±2.18	94.1 ±0.97	99.9 ±0.08	94.4 ±0.23	100.0 ±0.0	94.8 ±0.07	99.8 ±0.16	94.4 ±0.05
Bowl*	57.9 ±1.72	84.8 ±1.22	69.1 ±18.48	85.3 ±0.7	70.4 ±11.26	85.5 ±0.38	71.1 ±10.77	86.4 ±0.27
Mug	99.7 ±0.37	94.7 ±0.17	100.0 ±0.0	94.7 ±0.66	100.0 ±0.0	95.2 ±0.18	100.0 ±0.0	95.4 ±0.22
Power drill	95.0 ±3.39	92.2 ±1.15	96.8 ±3.61	92.9 ±1.11	96.2 ±1.52	91.8 ±0.47	96.5 ±1.33	92.6 ±0.2
Wood block*	94.2 ±6.19	87.8 ±1.41	98.5 ±0.78	87.4 ±1.16	95.5 ±1.79	87.5 ±0.71	94.4 ±3.45	87.7 ±1.24
Scissors	98.7 ±1.14	94.4 ±0.98	96.5 ±3.42	92.8 ±2.07	97.6 ±2.31	92.2 ±2.01	98.5 ±0.69	92.5 ±1.67
Marker	90.4 ±3.79	93.4 ±0.52	99.1 ±0.33	94.5 ±0.09	99.7 ±0.26	94.7 ±0.34	98.4 ±1.46	94.7 ±0.43
Clamp*	77.2 ±0.13	72.6 ±0.53	76.5 ±0.63	72.2 ±0.43	77.1 ±0.75	71.7 ±0.63	77.9 ±0.07	71.3 ±0.21
Large clamp*	69.7 ±0.84	69.0 ±0.17	71.1 ±0.78	72.2 ±1.68	70.7 ±0.37	73.6 ±0.2	72.8 ±0.85	72.6 ±1.76
Foam brick*	100.0 ±0.0	92.2 ±0.17	100.0 ±0.0	93.6 ±0.54	100.0 ±0.0	92.9 ±0.25	100.0 ±0.0	92.9 ±0.65
All objects	93.3 ±0.15	90.8 ±0.07	**94.7** ±0.08	**91.2** ±0.04	94.6 ±0.24	91.1 ±0.16	**94.7** ±0.63	**91.2** ±0.20

E.4. Details of Experiment 4 (Feature prediction): $\mathrm{SO}(3)$ as input to Fourier series

The network is given by a 3-layer MLP with input size being determined by the rotation representation, hidden dimension $[256, 256]$, ReLU activation functions, and the output size of one. The networks were trained for 400 Epochs using Adam (standard settings in PyTorch with a starting learning rate of 0.001). At each iteration a batch of 64 data points was drawn from 800 train data points and validated on 200 validation points. After training, the best model with respect to the validation loss was used to compute a test loss on 1000 data points. For train, validation, and test loss we used the RMSE.

E.5. Experiment 5 (Rotation estimation): Inverse Kinematics with CMU MoCap data

The works of Zhou et al. (2019); Levinson et al. (2020); Brégier (2021); Pepe et al. (2022) do rotation estimation for an Inverse Kinematics problem. In this setup, the model estimates joint rotations from the joint positions of a human pose as shown in Figure 19. These rotations are to be estimated relative to a canonical "t-pose", defined as a skeleton in a standing position with arms stretched. We refrained from reproducing this experiment as the dataset mostly contains angles close to the unit rotation as shown in Figure 19. This corresponds to the "small-angle" case as discussed in Section 4.1.

This plot was generated using 10,000 samples from the CMU MoCap database (LLC., 2024). The sampling process involved filtering out videos with 100 frames (static poses), randomly selecting 760 videos, and subsampling them to 30 frames each. From this collection of human poses, we uniformly sampled 10,000 poses. The joint rotations for each frame were then extracted relative to parent joints and expressed in relation to the "t-pose" joint rotations.

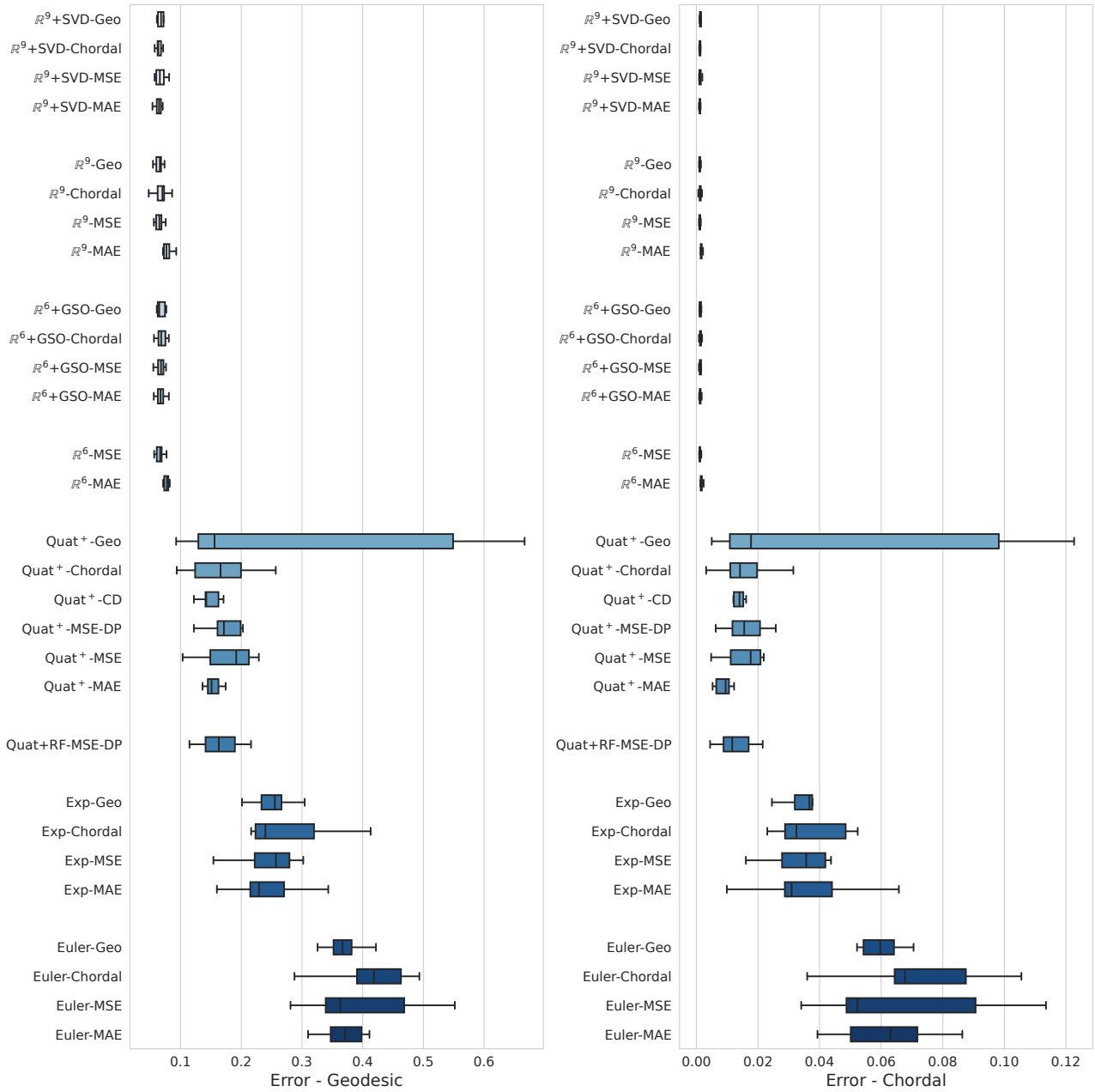

Figure 23. Experiment 1: Point clouds to rotation. Each box shows the test errors (Geodesic distance or Chordal distance) of ten networks trained on a specific rotation representation and loss function. For quaternions, the cosine distance '-CD' as in (4) and distance picking metric '-MSE-DP' as in (5) have also been used. "Quat-RF" refers to using quaternions which have been randomly multiplied by either -1 or 1. We only show "Quat-RF-MSE-DP" as using "Quat-RF" with CD, MAE, and MSE yielded errors four to seven times larger as the shown results. $\mathbb{R}^6$ and $\mathbb{R}^9$ refers to directly using the first two and three column vectors of the rotation matrix, respectively. Notably, resorting to the MAE can improve results for low-dimensional representations, as observed in the instances of quaternions, Euler angles, and exponential coordinates.

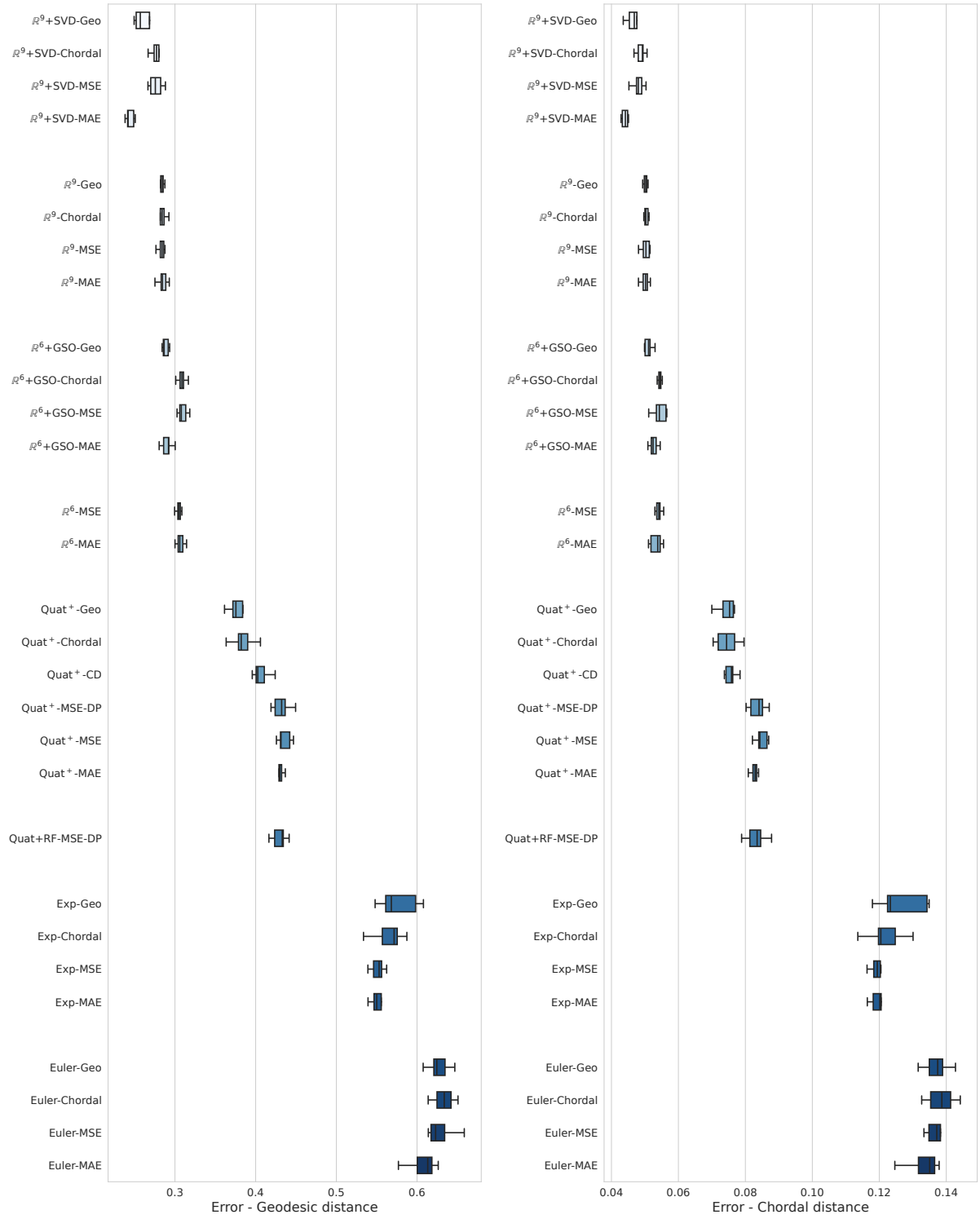

Figure 24. Experiment 2.1: Cube image to rotation. Each box shows the test errors (Geodesic distance or Chordal distance) of ten networks trained on a specific rotation representation and loss function. For quaternions, the cosine distance '-CD' as in (4) and distance picking metric '-MSE-DP' as in (5) have also been used. The observed relative performance aligns with previous experiments illustrated in Figure 23. "Quat-RF" refers to using quaternions which have been randomly multiplied by either -1 or 1. We only show "Quat-RF-MSE-DP" as using "Quat-RF" with CD, MAE, and MSE yielded errors four to six times larger as the shown results. $\mathbb{R}^6$ and $\mathbb{R}^9$ refers to directly using the first two and three column vectors of the rotation matrix, respectively.