Learning Equality Constraints for Motion Planning on Manifolds

Giovanni Sutanto*, Isabel M. Rayas Fernández, Peter Englert, Ragesh K. Ramachandran, Gaurav S. Sukhatme
Robotic Embedded Systems Laboratory
University of Southern California, United States
gsutanto@alumni.usc.edu
{rayas,penglert,rageshku,gaurav}@usc.edu

Abstract: Constrained robot motion planning is a widely used technique to solve complex robot tasks. We consider the problem of learning representations of constraints from demonstrations with a deep neural network, which we call Equality Constraint Manifold Neural Network (ECoMaNN). The key idea is to learn a level-set function of the constraint suitable for integration into a constrained sampling-based motion planner. Learning proceeds by aligning subspaces in the network with subspaces of the data. We combine both learned constraints and analytically described constraints into the planner and use a projection-based strategy to find valid points. We evaluate ECoMaNN on its representation capabilities of constraint manifolds, the impact of its individual loss terms, and the motions produced when incorporated into a planner.
Video: https://www.youtube.com/watch?v=WoC7nqp4XNk
Code: https://github.com/gsutanto/smp_manifold_learning

Keywords: manifold learning, motion planning, learning from demonstration

1 Introduction

Robots must be able to plan motions that follow various constraints in order for them to be useful in real-world environments. Constraints such as holding an object, maintaining an orientation, or staying within a certain distance of an object of interest are just some examples of possible restrictions on a robot's motion. In general, two approaches to many robotics problems can be described. One is the traditional approach of using handwritten models to capture environments, physics, and other aspects of the problem mathematically or analytically, and then solving or optimizing these to find a solution. The other, popularized more recently, involves the use of machine learning to replace, enhance, or simplify these hand-built parts. Both have challenges: Acquiring training data for learning can be difficult and expensive, while describing precise models analytically can range from tedious to impossible. Here, we approach the problem from a machine learning perspective and propose a solution to learn constraints from demonstrations. The learned constraints can be used alongside analytical solutions within a motion planning framework.

In this work, we propose a new learning-based method for describing motion constraints, called Equality Constraint Manifold Neural Network (ECoMaNN). ECoMaNN learns a function which evaluates a robot configuration on whether or not it meets the constraint, and for configurations near the constraint, on how far away it is. We train ECoMaNN with datasets consisting of configurations that adhere to constraints, and present results for kinematic robot tasks learned from demonstrations. We use a sequential motion planning framework to solve motion planning problems that are both constrained and sequential in nature, and incorporate the learned constraint representations into it. We evaluate the constraints learned by ECoMaNN with various datasets on their representation quality. Further, we investigate the usability of learned constraints in sequential motion planning problems.

* Giovanni Sutanto is now at X Development LLC. He contributed to this work during his past affiliation with the Robotic Embedded Systems Laboratory at USC.

2 Related work

2.1 Manifold learning

Manifold learning is applicable to many fields and thus there exist a wide variety of methods for it. Linear methods include PCA and LDA [1], and while they are simple, they lack the complexity to represent complex manifolds. Nonlinear methods include multidimensional scaling (MDS), locally linear embedding (LLE), Isomap, and local tangent space alignment (LTSA). These approaches use techniques such as eigenvalue decomposition, nearest neighbor reconstructions, and local-structure-preserving graphs to visualize and represent manifolds. In LTSA, the local tangent space information of each point is aligned to create a global representation of the manifold. We refer the reader to [2] for details. Recent work in manifold learning additionally takes advantage of (deep) neural architectures. Some approaches use autoencoder-like models [3, 4] or deep neural networks [5] to learn manifolds, e.g. of human motion. Others use classical methods combined with neural networks, for example as a loss function for control [6] or as structure for the network [7]. Locally Smooth Manifold Learning (LSML) [8] defines and learns a function which describes the tangent space of the manifold, allowing randomly sampled points to be projected onto it. Our work is related to many of these approaches; in particular, the tangent space alignment in LTSA is an idea that ECoMaNN uses extensively. Similar to the ideas presented in this paper, the work in [9] delineates an approach to solve motion planning problems by learning the solution manifold of an optimization problem. In contrast to others, our work focuses on learning implicit functions of equality constraint manifolds, which is a generalization of the learning representations for Signed Distance Fields (SDF) [10], up to a scale, for higher-dimensional manifolds.

2.2 Learning from demonstration

Learning from demonstration (LfD) techniques learn a task representation from data which is usable to generate robot motions that imitate the demonstrated behavior. One approach to LfD is inverse optimal control (IOC), which aims to find a cost function that describes the demonstrated behavior [11, 12, 13]. Recently, IOC has been extended to extract constraints from demonstrations [14, 15]. There, a cost function as well as equality and inequality constraints are extracted from demonstrations, which are useful to describe behavior like contacts or collision avoidance. Our work can be seen as a special case where the task is only represented in form of constraints. Instead of using the extracted constraints in optimal control methods, we integrate them into sampling-based motion planning methods, which are not parameterized by time and do not suffer from poor initializations. A more direct approach to LfD is to learn parameterized motion representations [16, 17, 18]. They represent the demonstrations in a parameterized form such as Dynamic Movement Primitives [19]. Here, learning a primitive from demonstration is often possible via linear regression; however, the ability to generalize to new situations is more limited. Other approaches to LfD include task space learning [20] and deep learning [21]. We refer the reader to the survey [22] for a broad overview on LfD.

2.3 Constrained sampling-based motion planning

Sampling-based motion planning (SBMP) is a broad field which tackles the problem of motion planning by using randomized sampling techniques to build a tree or graph of configurations (also called samples), which can then be used to plan paths between configurations. Many SBMP algorithms derive from rapidly-exploring random trees (RRT) [23], probabilistic roadmaps (PRM) [24], or their optimal counterparts [25]. A more challenging and realistic motion planning task is that of constrained SBMP [26], where there are motion constraints beyond just obstacle avoidance which lead to a free configuration space manifold of lower dimension than the ambient configuration space. Previous research has also investigated incorporating learned constraints or manifolds into planning frameworks. These include performing planning in learned latent spaces [27], learning a better sampling distribution in order to take advantage of the structure of valid configurations rather than blindly sample uniformly in the search space [28, 29], and attempting to approximate the manifold (both explicitly and implicitly) of valid points with graphs in order to plan on them more effectively [30, 31, 32]. Our method differs from previous work in that ECoMaNN learns an implicit description of a constraint manifold via a level set function, and during planning, we assume this representation for each task. We note that our method could be combined with others, e.g. learned sampling distributions, to further improve planning results.

3 Background

3.1 Manifold theory

Here we present the necessary background on manifold theory [33]. Informally, a manifold is a surface which can be well-approximated locally using an open set of a Euclidean space near every point. Manifolds are generally defined using *charts*, which are collections of open sets whose union yields the manifold, and a *coordinate map*, which is a continuous map associated with each set. However, an alternative representation which is useful from a computational perspective is to represent the manifold as the zero level set of a continuous function. Since the latter representation is a direct result of the implicit function theorem, it is referred to as *implicit representation* of the manifold. For example, the manifold represented by the zero level set of the function $h_M(x, y) = x^2 + y^2 - 1$ (i.e. $\{(x, y) \mid x^2 + y^2 - 1 = 0\}$) is a circle. Moreover, the implicit function associated with a smooth manifold is smooth. Thus, we can associate a manifold with every equality constraint. The vector space containing the set of all tangent vectors at $\mathbf{q}$ is denoted using $T_{\mathbf{q}}M$. Given a manifold with the corresponding implicit function $h_M(\mathbf{q})$, if we endow its tangent spaces with an appropriate inner product structure, then such a manifold is often referred as a Riemannian manifold. In this work the manifolds are assumed to be Riemannian.

3.2 Motion planning on manifolds

In this work, we aim to integrate learned constraint manifolds into a motion planning framework [34]. The motion planner considers kinematic motion planning problems in a configuration space $\mathcal{C} \subseteq \mathbb{R}^d$. A robot configuration $\mathbf{q} \in \mathcal{C}$ describes the state of one or more robots with d degrees of freedom in total. A manifold M is represented as an equality constraint $h_M(\mathbf{q}) = \mathbf{0}$. The set of robot configurations that are on a manifold M is given by $\mathcal{C}_M = \{\mathbf{q} \in \mathcal{C} \mid h_M(\mathbf{q}) = \mathbf{0}\}$. The planning problem is defined as a sequence of $(m + 1)$ such manifolds $\mathcal{M} = \{M_1, M_2, \ldots, M_{m+1}\}$ and an initial configuration $\mathbf{q}_{\text{start}} \in \mathcal{C}_{M_1}$ on the first manifold. The goal is to find a path from $\mathbf{q}_{\text{start}}$ that traverses the manifold sequence $\mathcal{M}$ and reaches a configuration on the goal manifold M_{m+1}. A path on the i-th manifold is defined as $\tau_i : [0, 1] \to \mathcal{C}_{M_i}$ and $J(\tau_i)$ is the cost function of a path $J : \mathcal{T} \to \mathbb{R}_{\geq 0}$ where $\mathcal{T}$ is the set of all non-trivial paths. The problem is formulated as an optimization over a set of paths $\boldsymbol{\tau} = (\tau_1, \ldots, \tau_m)$ that minimizes the sum of path costs under the constraints of traversing $\mathcal{M}$:

$$\begin{aligned} \boldsymbol{\tau}^\star &= \arg\min_{\boldsymbol{\tau}} \sum_{i=1}^{m} J(\tau_i) \\ \text{s.t.} \quad & \tau_1(0) = \mathbf{q}_{\text{start}}, \quad \tau_m(1) \in \mathcal{C}_{M_{m+1}} \cap \mathcal{C}_{\text{free},m+1}, \quad \tau_i(1) = \tau_{i+1}(0) \quad \forall_{i=1,\ldots,m-1} \\ & \mathcal{C}_{\text{free},i+1} = \Upsilon(\mathcal{C}_{\text{free},i}, \tau_i) \quad \forall_{i=1,\ldots,m}, \quad \tau_i(s) \in \mathcal{C}_{M_i} \cap \mathcal{C}_{\text{free},i} \quad \forall_{i=1,\ldots,m} \forall_{s\in[0,1]} \end{aligned} \tag{1}$$

Υ is an operator that describes the change in the free configuration space (the space of all configurations that are not in collision with the environment) $\mathcal{C}_{\text{free}}$ when transitioning to the next manifold. The operator Υ is not explicitly known and we only assume to have access to a collision checker that depends on the current robot configuration and the object locations in the environment. Intelligently performing goal-achieving manipulations that change the free configuration space forms a key challenge in robot manipulation planning. The SMP* (Sequential Manifold Planning) algorithm is able to solve this problem for a certain class of motion planning scenarios. It iteratively applies RRT* to find a path that reaches the next manifold while staying on the current manifold. For further details of the SMP* algorithm, we refer the reader to [34]. In this paper, we employ data-driven learning methods to learn individual equality constraints $h_M(\mathbf{q}) = 0$ from demonstrations with the goal to integrate them with analytically defined manifolds into this framework.

4 Equality Constraint Manifold Neural Network (ECoMaNN)

We propose a novel neural network structure, called *Equality Constraint Manifold Neural Network* (ECoMaNN), which is a (global) equality constraint manifold learning representation that enforces the alignment of the (local) tangent spaces and normal spaces with the information extracted from the Local Principal Component Analysis (Local PCA) [35] of the data. ECoMaNN takes a configuration $\mathbf{q}$ as input and outputs the prediction of the implicit function $h_M(\mathbf{q})$. We train ECoMaNN in a supervised manner, from demonstrations. One of the challenges is that the supervised training dataset is collected only from demonstrations of data points which are on the manifold $\mathcal{C}_M$, called the *on-manifold* dataset. Collecting both the on-manifold $\mathcal{C}_M$ and off-manifold $\mathcal{C}_{\setminus M} = \{\mathbf{q} \in \mathcal{C} \mid h_M(\mathbf{q}) \neq \mathbf{0}\}$ datasets for supervised training would be tedious because the implicit function h_M values of points in $\mathcal{C}_{\setminus M}$ are typically unknown and hard to label. We show that, though our approach is only given data on $\mathcal{C}_M$, it can still learn a useful and sufficient representation of the manifold for use in planning.

Our goal is to learn a single global representation of the constraint manifold M in the form of a neural network. Our approach leverages local information on the manifold in the form of the tangent and normal spaces [36]. With regard to h_M, the tangent and normal spaces are equivalent to the null and row space, respectively, of the Jacobian matrix $\mathbf{J}_M(\acute{\mathbf{q}}) = \left.\frac{\partial h_M(\mathbf{q})}{\partial \mathbf{q}}\right|_{\mathbf{q}=\acute{\mathbf{q}}}$, and valid in a small neighborhood around the point $\acute{\mathbf{q}}$. Using on-manifold data, the local information of the manifold can be analyzed using Local PCA. For each data point $\mathbf{q}$ in the on-manifold dataset, we establish a local neighborhood using K-nearest neighbors (KNN) $\hat{\mathcal{K}} = \{\hat{\mathbf{q}}_1, \hat{\mathbf{q}}_2, \ldots, \hat{\mathbf{q}}_K\}$, with $K \geq d$. After a change of coordinates, $\mathbf{q}$ becomes the origin of a new local coordinate frame $\mathcal{F}_L$, and the KNN becomes $\tilde{\mathcal{K}} = \{\tilde{\mathbf{q}}_1, \tilde{\mathbf{q}}_2, \ldots, \tilde{\mathbf{q}}_K\}$ with $\tilde{\mathbf{q}}_k = \hat{\mathbf{q}}_k - \mathbf{q}$ for all values of k. Defining the matrix $\mathbf{X} = \begin{bmatrix}\tilde{\mathbf{q}}_1 & \tilde{\mathbf{q}}_2 & \ldots & \tilde{\mathbf{q}}_K\end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{K \times d}$, we can compute the covariance matrix $\mathbf{S} = \frac{1}{K-1}\mathbf{X}^{\mathrm{T}}\mathbf{X} \in \mathbb{R}^{d \times d}$. The eigendecomposition of $\mathbf{S} = \mathbf{V}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$ gives us the Local PCA. The matrix $\mathbf{V}$ contains the eigenvectors of $\mathbf{S}$ as its columns in decreasing order w.r.t. the corresponding eigenvalues in the diagonal matrix $\mathbf{\Sigma}$. These eigenvectors form the basis of the coordinate frame $\mathcal{F}_L$.

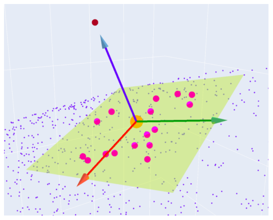


Figure 1: A visualization of data augmentation along the 1D normal space of a point $\mathbf{q}$ in 3D space. Here, purple points are the dataset, pink points are the KNN of $\mathbf{q}$, and the dark red point is $\check{\mathbf{q}}$. $\mathbf{q}$ is at the axes origin, and the green plane is the approximated tangent space at that point.

This local coordinate frame $\mathcal{F}_L$ is tightly related to the tangent space $T_{\mathbf{q}}M$ and the normal space $N_{\mathbf{q}}M$ of the manifold M at $\mathbf{q}$. That is, the $(d-l)$ eigenvectors corresponding to the $(d-l)$ biggest eigenvalues of $\mathbf{\Sigma}$ form a basis of $T_{\mathbf{q}}M$, while the remaining l eigenvectors form the basis of $N_{\mathbf{q}}M$. Furthermore, the l smallest eigenvalues of $\mathbf{\Sigma}$ will be close to zero, resulting in the l eigenvectors associated with them forming the basis of the null space of $\mathbf{S}$. On the other hand, the remaining $(d-l)$ eigenvectors form the basis of the row space of $\mathbf{S}$. We follow the same technique as Deutsch and Medioni [36] for automatically determining the number of constraints l from data, which is also the number of outputs of ECoMaNN[1]. Suppose the eigenvalues of $\mathbf{S}$ are $\{\lambda_1, \lambda_2, \ldots, \lambda_d\}$ (in decreasing order w.r.t. magnitude). Then the number of constraints can be determined as $l = \arg\max\left([\lambda_1 - \lambda_2, \lambda_2 - \lambda_3, \ldots, \lambda_{d-1} - \lambda_d]\right)$.

We now present several methods to define and train ECoMaNN, as follows:

4.1 Alignment of local tangent and normal spaces

ECoMaNN aims to align the following:

(a) the null space of $\mathbf{J}_M$ and the row space of $\mathbf{S}$ (which must be equivalent to tangent space $T_{\mathbf{q}}M$)
(b) the row space of $\mathbf{J}_M$ and the null space of $\mathbf{S}$ (which must be equivalent to normal space $N_{\mathbf{q}}M$)

for the local neighborhood of each point $\mathbf{q}$ in the on-manifold dataset. Suppose the eigenvectors of $\mathbf{S}$ are $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_d\}$ and the singular vectors of $\mathbf{J}_M$ are $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_d\}$, where the indices indicate decreasing order w.r.t. the eigenvalue/singular value magnitude. The null spaces of $\mathbf{S}$ and $\mathbf{J}_M$ are spanned by $\{\boldsymbol{v}_{d-l+1}, \ldots, \boldsymbol{v}_d\}$ and $\{\boldsymbol{e}_{l+1}, \ldots, \boldsymbol{e}_d\}$, respectively. The two conditions above imply that the projection of the null space eigenvectors of $\mathbf{J}_M$ into the null space of $\mathbf{S}$ should be $\mathbf{0}$, and similarly for the row spaces. Hence, we achieve this by training ECoMaNN to minimize projection errors $\left\|\mathbf{V}_{\mathrm{N}}\mathbf{V}_{\mathrm{N}}^{\mathrm{T}}\mathbf{E}_{\mathrm{N}}\right\|_2^2$ and $\left\|\mathbf{E}_{\mathrm{N}}\mathbf{E}_{\mathrm{N}}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}}\right\|_2^2$ with $\mathbf{V}_{\mathrm{N}} = \begin{bmatrix}\boldsymbol{v}_{d-l+1} & \ldots & \boldsymbol{v}_d\end{bmatrix}$ and $\mathbf{E}_{\mathrm{N}} = \begin{bmatrix}\boldsymbol{e}_{l+1} & \ldots & \boldsymbol{e}_d\end{bmatrix}$ at all the points in the on-manifold dataset.

[1]Here we assume that the intrinsic dimensionality of the manifold at each point remains constant.

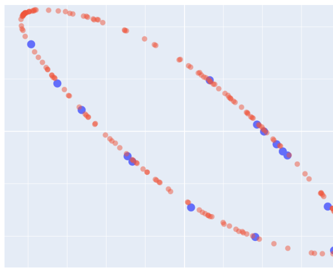
4.2 Data augmentation with off-manifold data

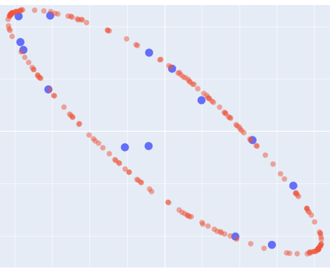
The training dataset is on-manifold, i.e., each point $\mathbf{q}$ in the dataset satisfies $h_M(\mathbf{q}) = \mathbf{0}$. Through Local PCA on each of these points, we know the data-driven approximation of the normal space of the manifold at $\mathbf{q}$. Hence, we know the directions where the violation of the equality constraint increases, i.e., the same direction as any vector picked from the approximate normal space. Since our future use of the learned constraint manifold on motion planning does not require the acquisition of the near-ground-truth value of $h_M(\mathbf{q}) \neq \mathbf{0}$, we can set this off-manifold valuation of h_M arbitrarily, as long as it does not interfere with the utility for projecting an off-manifold point onto the manifold. Therefore, we can augment our dataset with off-manifold data to achieve a more robust learning of ECoMaNN. For each point $\mathbf{q}$ in the on-manifold dataset, and for each random unit vector $\boldsymbol{u}$ picked from the normal space at $\mathbf{q}$, we can add an off-manifold point $\check{\mathbf{q}} = \mathbf{q} + i\epsilon\boldsymbol{u}$ with a positive integer i and a small positive scalar ϵ (see Figure 1 for a visualization). However, if the closest on-manifold data point to an augmented point $\check{\mathbf{q}} = \mathbf{q} + i\epsilon\boldsymbol{u}$ is not $\mathbf{q}$, we reject it. This prevents situations like augmenting a point on a sphere beyond the center of the sphere. Data augmentation is a technique used in various fields, and our approach has similarities to others [37, 38], though in this work we focus on using augmentation to aid learning an implicit constraint function for robotic motion planning. With this data augmentation, we now define several losses used to train ECoMaNN.

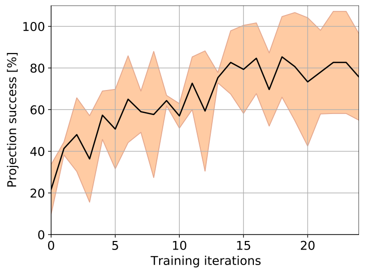
4.2.1 Training losses

Loss based on the norm of the output vector of ECoMaNN For the augmented data point $\check{\mathbf{q}} = \mathbf{q} + i\epsilon\boldsymbol{u}$, we set the label satisfying $\|h_M(\check{\mathbf{q}})\|_2 = i\epsilon$. During training, we minimize the norm prediction error $\mathcal{L}_{\text{norm}} = \|(\|h_M(\check{\mathbf{q}})\|_2 - i\epsilon)\|_2^2$ for each augmented point $\check{\mathbf{q}}$.
Furthermore, we define the following three siamese losses. The main intuition behind these losses is that we expect the learned function h_M to output similar values for similar points.
Siamese loss for reflection pairs For the augmented data point $\check{\mathbf{q}} = \mathbf{q} + i\epsilon\boldsymbol{u}$ and its reflection pair $\mathbf{q} - i\epsilon\boldsymbol{u}$, we can expect that $h_M(\mathbf{q} + i\epsilon\boldsymbol{u}) = -h_M(\mathbf{q} - i\epsilon\boldsymbol{u})$, or in other words, that an augmented point and its reflection pair should have the same h_M but with opposite signs. Therefore, during training we minimize the siamese loss $\mathcal{L}_{\text{reflection}} = \|h_M(\mathbf{q} + i\epsilon\boldsymbol{u}) + h_M(\mathbf{q} - i\epsilon\boldsymbol{u})\|_2^2$.
Siamese loss for augmentation fraction pairs Similarly, between the pair $\check{\mathbf{q}} = \mathbf{q} + i\epsilon\boldsymbol{u}$ and $\mathbf{q} + \frac{a}{b} i\epsilon\boldsymbol{u}$, where a and b are positive integers satisfying $0 < \frac{a}{b} < 1$, we can expect that $\frac{h_M(\mathbf{q}+i\epsilon\boldsymbol{u})}{\|h_M(\mathbf{q}+i\epsilon\boldsymbol{u})\|_2} = \frac{h_M(\mathbf{q}+\frac{a}{b}i\epsilon\boldsymbol{u})}{\left\|h_M(\mathbf{q}+\frac{a}{b}i\epsilon\boldsymbol{u})\right\|_2}$. In other words, the *normalized* h_M values should be the same on an augmented point $\mathbf{q} + i\epsilon\boldsymbol{u}$ as on any point in between the on-manifold point $\mathbf{q}$ and that augmented point $\mathbf{q} + i\epsilon\boldsymbol{u}$. Hence, during training we minimize the siamese loss $\mathcal{L}_{\text{fraction}} = \left\| \frac{h_M(\mathbf{q}+i\epsilon\boldsymbol{u})}{\|h_M(\mathbf{q}+i\epsilon\boldsymbol{u})\|_2} - \frac{h_M(\mathbf{q}+\frac{a}{b}i\epsilon\boldsymbol{u})}{\left\|h_M(\mathbf{q}+\frac{a}{b}i\epsilon\boldsymbol{u})\right\|_2} \right\|_2^2$.
Siamese loss for similar augmentation pairs Suppose for nearby on-manifold data points $\mathbf{q}_a$ and $\mathbf{q}_c$, their approximate normal spaces $N_{\mathbf{q}_a}M$ and $N_{\mathbf{q}_c}M$ are spanned by eigenvector bases $\mathcal{F}_N^a = \{\boldsymbol{v}_{d-l+1}^a, \ldots, \boldsymbol{v}_d^a\}$ and $\mathcal{F}_N^c = \{\boldsymbol{v}_{d-l+1}^c, \ldots, \boldsymbol{v}_d^c\}$, respectively. If $\mathcal{F}_N^a$ and $\mathcal{F}_N^c$ are closely aligned, the random unit vectors $\boldsymbol{u}_a$ from $\mathcal{F}_N^a$ and $\boldsymbol{u}_c$ from $\mathcal{F}_N^c$ can be obtained by $\boldsymbol{u}_a = \frac{\sum_{j=d-l+1}^{d} w_j \boldsymbol{v}_j^a}{\left\|\sum_{j=d-l+1}^{d} w_j \boldsymbol{v}_j^a\right\|_2}$ and $\boldsymbol{u}_c = \frac{\sum_{j=d-l+1}^{d} w_j \boldsymbol{v}_j^c}{\left\|\sum_{j=d-l+1}^{d} w_j \boldsymbol{v}_j^c\right\|_2}$, where $\{w_{d-l+1}, \ldots, w_d\}$ are random scalar weights from a standard normal distribution common to both the bases of $\mathcal{F}_N^a$ and $\mathcal{F}_N^c$. This will ensure that $\boldsymbol{u}_a$ and $\boldsymbol{u}_c$ are aligned as well, and we can expect that $h_M(\mathbf{q}_a + i\epsilon\boldsymbol{u}_a) = h_M(\mathbf{q}_c + i\epsilon\boldsymbol{u}_c)$. In other words, two aligned augmented points in the same level set should have the same h_M value. Therefore, during training we minimize the siamese loss $\mathcal{L}_{\text{similar}} = \|h_M(\mathbf{q}_a + i\epsilon\boldsymbol{u}_a) - h_M(\mathbf{q}_c + i\epsilon\boldsymbol{u}_c)\|_2^2$. In general, the alignment of $\mathcal{F}_N^a$ and $\mathcal{F}_N^c$ is not guaranteed, for example due to the numerical sensitivity of singular value/eigen decomposition. Therefore, we introduce an algorithm for Orthogonal Subspace Alignment (OSA) in the Supplementary Material to ensure that this assumption is satisfied.

While $\mathcal{L}_{\text{norm}}$ governs only the norm of ECoMaNN's output, the other three losses $\mathcal{L}_{\text{reflection}}$, $\mathcal{L}_{\text{fraction}}$, and $\mathcal{L}_{\text{similar}}$ constrain the (vector) outputs of ECoMaNN based on pairwise input data points without explicitly hand-coding the desired output itself. We avoid the hand-coding of the desired output because this is difficult for high-dimensional manifolds, except when there is prior knowledge about the manifold available, such as in the case of Signed Distance Fields (SDF) manifolds.


(a) Samples from ECoMaNN


(b) Samples from VAE



(c) ECoMaNN projection success

Figure 2: Images a and b visualize a slice near $z = 0$ of the Plane dataset for experiment 5.1. Red points are the training dataset and blue points are samples generated from the learned manifolds. The points projected onto the manifold using ECoMaNN are closer to the manifold, with an 85% projection success rate. A significant portion of the points generated using the VAE lie inside the surface, which leads to a lower success rate of 77%. Figure c shows the projection success of ECoMaNN over the number of training iterations. The quantitative results are found in Table 1.

Table 1: Accuracy and precision of learned manifolds averaged across 3 instances. "Train" indicates results on the on-manifold training set; "test" indicates $N = 1000$ projected (ECoMaNN) or sampled (VAE) points.

	ECoMaNN			VAE		
Dataset	$\mu_{\bar{h}_M}$ (train)	$\mu_{\bar{h}_M}$ (test)	$P_{\bar{h}_M}$	$\mu_{\bar{h}_M}$ (train)	$\mu_{\bar{h}_M}$ (test)	$P_{\bar{h}_M}$
Sphere	0.024 ± 0.009	0.023 ± 0.009	100.0 ± 0.0	0.105 ± 0.088	0.161 ± 0.165	46.867 ± 18.008
3D Circle	0.029 ± 0.011	0.030 ± 0.011	78.0 ± 22.0	0.894 ± 0.074	0.902 ± 0.069	0.0 ± 0.0
Plane	0.020 ± 0.005	0.020 ± 0.005	88.5 ± 10.5	0.053 ± 0.075	0.112 ± 0.216	77.733 ± 7.721
Orient	0.090 ± 0.009	0.090 ± 0.009	73.5 ± 6.5	0.010 ± 0.037	0.085 ± 0.237	85.9 ± 1.068

5 Experiments

We use the robot simulator MuJoCo [39] to generate four datasets. The size of each dataset is denoted as N. We define a ground truth constraint $\bar{h}_M$, randomly sample points in the configuration (joint) space, and use a constrained motion planner to find robot configurations in $\mathcal{C}_M$ that produce the on-manifold datasets: **Sphere**: 3D point that has to stay on the surface of a sphere. $N = 5000, d = 3, l = 1$. **3D Circle**: A point that has to stay on a circle in 3D space. $N = 1000, d = 3, l = 2$. **Plane**: Robot arm with 3 rotational DoFs where the end effector has to be on a plane. $N = 20000, d = 3, l = 1$. **Orient**: Robot arm with 6 rotational DoFs that has to keep its orientation upright (e.g., transporting a cup). $N = 21153, d = 6, l = 2$. In the following experiments, we parametrize ECoMaNN with 4 hidden layers of size 36, 24, 18, and 10. The hidden layers use a $\tanh$ activation function and the output layer is linear.

5.1 Accuracy and precision of learned manifolds

We compare the accuracy and precision of the manifolds learned by ECoMaNN with those learned by a variational autoencoder (VAE) [40]. VAEs are a popular generative model that embeds data points as a distribution in a learned latent space, and as such new latent vectors can be sampled and decoded into new examples which fit the distribution of the training data.[2] We use two metrics: First, the distance $\mu_{\bar{h}_M}$ which measures how far a point is away from the ground-truth manifold $\bar{h}_M$ and which we evaluate for both the training data points and randomly sampled points, and second, the percent $P_{\bar{h}_M}$ of randomly sampled points that are on the manifold $\bar{h}_M$. We use a distance threshold of 0.1 to determine success when calculating $P_{\bar{h}_M}$. For ECoMaNN, randomly sampled points are projected using gradient descent with the learned implicit function until convergence. For the VAE, latent points are sampled from $\mathcal{N}(0, 1)$ and decoded into new configurations.

[2]We also tested classical manifold learning techniques (Isomap, LTSA, PCA, MDS, and LLE). We found them empirically not expressive enough and/or unable to support projection or sampling operations, necessary capabilities for this work.

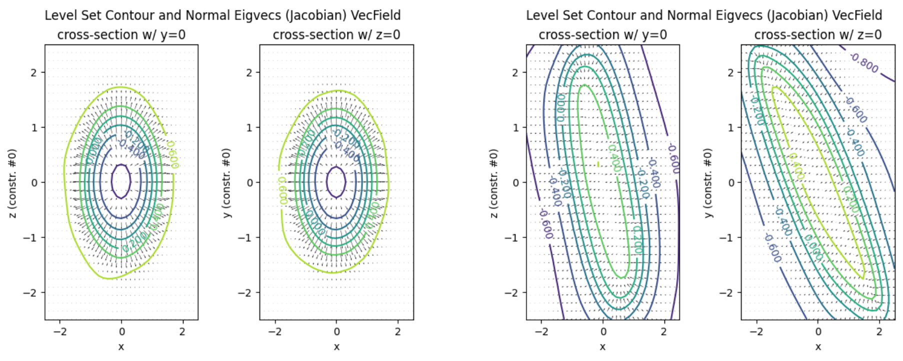


Figure 3: Trained ECoMaNN's level set contour plot and the normal space eigenvector field, after training on the sphere constraint dataset (left) and plane constraint dataset (right).

Table 2: Percentage of projection success rate for a variety of ablations of ECoMaNN components.

Ablation Type	Sphere	3D Circle	Plane
No Ablation	(98.67 ± 1.89) %	(100.00 ± 0.00) %	(92.33 ± 10.84) %
w/o Data Augmentation	(9.00 ± 2.45) %	(16.67 ± 4.64) %	(3.67 ± 3.30) %
w/o OSA	(64.33 ± 9.03) %	(33.33 ± 17.13) %	(61.00 ± 6.16) %
w/o Siamese Losses	(17.33 ± 3.40) %	(65.67 ± 26.03) %	(35.67 ± 5.56) %
w/o Siamese Loss $\mathcal{L}_{\text{reflection}}$	(92.67 ± 4.03) %	(9.67 ± 2.05) %	(38.00 ± 16.87) %
w/o Siamese Loss $\mathcal{L}_{\text{fraction}}$	(88.33 ± 8.34) %	(99.67 ± 0.47) %	(85.33 ± 16.68) %
w/o Siamese Loss $\mathcal{L}_{\text{similar}}$	(83.00 ± 5.10) %	(70.67 ± 21.00) %	(64.33 ± 2.62) %

We sample 1000 points for each of these comparisons. We report results in Table 1 and a visualization of projected samples in Fig. 2. We also plot the level set and the normal space eigenvector field of the ECoMaNN trained on the sphere and plane constraint dataset in Fig. 3. In all experiments, we set the value of the augmentation magnitude ϵ to the square root of the mean eigenvalues of the approximate tangent space, which we found to work well experimentally. With the exceptions of the embedding size and the input size, which are set to the dimensionality $d - l$ as the tangent space of the constraint learned by ECoMaNN and the ambient space dimensionality d of the dataset, respectively, the VAE has the same parameters for each dataset: 4 hidden layers with 128, 64, 32, and 16 units in the encoder and the same but reversed in the decoder; the weight of the KL divergence loss $\beta = 0.01$; using batch normalization; and trained for 100 epochs.

Our results show that for every dataset except Orient, ECoMaNN out performs the VAE in both metrics. ECoMaNN additionally outperforms the VAE with the Orient dataset in the testing phase, which suggests more robustness of the learned model. We find that though the VAE also performs relatively well in most cases, it cannot learn a good representation of the 3D Circle constraint and fails to produce any valid sampled points. ECoMaNN, on the other hand, can learn to represent all four constraints well.

5.2 Ablation study of ECoMaNN

In the ablation study, we compare $P_{\bar{h}_M}$ across 7 different ECoMaNN setups: 1) no ablation; 2) without data augmentation; 3) without orthogonal subspace alignment (OSA) during data augmentation; 4) without siamese losses during training; 5) without $\mathcal{L}_{\text{reflection}}$; 6) without $\mathcal{L}_{\text{fraction}}$; and 7) without $\mathcal{L}_{\text{similar}}$. Results are reported in Table 2. The data suggest that all parts of the training process are essential for a high success rate during projection. Of the features tested, data augmentation appears to have the most impact. This makes sense because without augmented data to train on, any configuration that does not already lie on the manifold will have undefined output when evaluated with ECoMaNN. Additionally, results from ablating the individual siamese losses suggest that the contribution of each is dependent on the context and structure of the constraint. Complementary to this ablation study, we present some additional experimental results in the Supplementary Material.

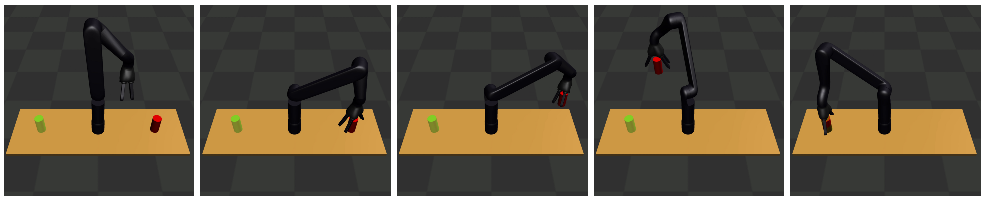


Figure 5: The images visualize a path that was planned on a learned orientation manifold.

5.3 Motion planning on learned manifolds

In the final experiment, we integrate ECoMaNN into the sequential motion planning framework described in Section 3.2. We mix the learned constraints with analytically defined constraints and evaluate it for two tasks. The first one is a geometric task, visualized in Figure 4, where a point starting on a paraboloid in 3D space must find a path to a goal state on another paraboloid. The paraboloids are connected by a sphere, and the point is constrained to stay on the surfaces at all times. In this case, we give the paraboloids analytically to the planner, and use ECoMaNN to learn the connecting constraint using the Sphere dataset. Figure 4 shows the resulting path where the sphere is represented by a learned manifold (red line) and where it is represented by the ground-truth manifold (black line). While the paths do not match exactly, both paths are on the manifold and lead to similar solutions in terms of path lengths. The task was solved in 27.09 s on a 2.2 GHz Intel Core i7 processor. The tree explored 1117 nodes and the found path consists of 24 nodes.

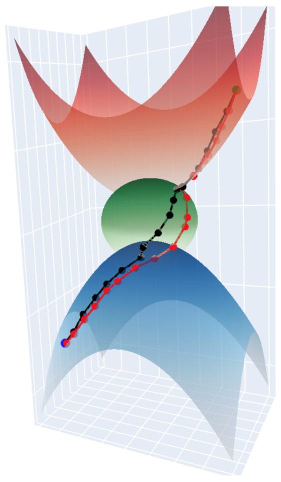


Figure 4: Planned path on the learned manifold (red) and on the ground truth manifold (black).

The second task is a robot pick-and-place task with the additional constraint that the transported object needs to be oriented upwards throughout the whole motion. For this, we use the Orient dataset to learn the manifold for the transport phase and combine it with other manifolds that describe the pick and place operation. The planning time was 42.97 s, the tree contained 1421 nodes and the optimal path had 22 nodes. Images of the resulting path are shown in Figure 5.

6 Discussion and conclusion

In this paper, we presented a novel method called Equality Constraint Manifold Neural Network for learning equality constraint manifolds from data. ECoMaNN works by aligning the row and null spaces of the local PCA and network Jacobian, which results in approximate learned normal and tangent spaces of the underlying manifold, suitable for use within a constrained sampling-based motion planner. In addition, we introduced a method for augmenting a purely on-manifold dataset to include off-manifold points and several loss functions for training. This improves the robustness of the learned method while avoiding hand-coding the labels for the augmented points. We also showed that the learned manifolds can be used in a sequential motion planning framework for constrained robot tasks.

While our experiments show success in learning a variety of manifolds, there are some limitations to our method. First, ECoMaNN by design can only learn equality constraints. Although many tasks can be specified with such constraints, inequality constraints are also an important part of many robot planning problems. Additionally, because of inherent limitations in learning from data, ECoMaNN does not guarantee that a randomly sampled point in configuration space will be projected successfully onto the learned manifold. This presents challenges in designing asymptotically complete motion planning algorithms, and is an important area of research. In the future, we plan on further testing ECoMaNN on more complex tasks, and in particular on tasks which are demonstrated by a human rather than from simulation.

Acknowledgments
This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1842487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work was supported in part by the Office of Naval Research (ONR) under grant N000141512550.

References

[1] R. O. Duda, P. E. Hart, and D. G. Stork. Pattern classification. *John Wiley & Sons, Inc.*, 2001.

[2] Y. Ma and Y. Fu. *Manifold learning theory and applications*. CRC press, 2011.

[3] D. Holden, J. Saito, T. Komura, and T. Joyce. Learning motion manifolds with convolutional autoencoders. In *ACM SIGGRAPH Asia 2015 Technical Briefs*, 2015.

[4] N. Chen, M. Karl, and P. van der Smagt. Dynamic movement primitives in latent space of time-dependent variational autoencoders. In *IEEE-RAS 16th Humanoids*, pages 629–636, 2016.

[5] X. S. Nguyen, L. Brun, O. Lézoray, and S. Bougleux. A neural network based on spd manifold learning for skeleton-based hand gesture recognition. In *IEEE CVPR*, 2019.

[6] G. Sutanto, N. Ratliff, B. Sundaralingam, Y. Chebotar, Z. Su, A. Handa, and D. Fox. Learning latent space dynamics for tactile servoing. In *IEEE ICRA*, pages 3622–3628, 2019.

[7] W. Wang, Y. Huang, Y. Wang, and L. Wang. Generalized autoencoder: A neural network framework for dimensionality reduction. In *IEEE CVPR Workshops*, pages 490–497, 2014.

[8] P. Dollár, V. Rabaud, and S. Belongie. Non-isometric manifold learning: Analysis and an algorithm. In *24th ICML*, pages 241–248, 2007.

[9] T. Osa. Learning the solution manifold in optimization and its application in motion planning. *arXiv*, 2020.

[10] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE CVPR*, 2019.

[11] N. D. Ratliff, J. A. Bagnell, and M. Zinkevich. Maximum margin planning. In *ICML*, 2006.

[12] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum Entropy Inverse Reinforcement Learning. *In Proceedings of the AAAI Conference on Artificial Intelligence*, 2008.

[13] S. Levine and V. Koltun. Continuous inverse Optimal Control with Locally Optimal Examples. In *ICML*, 2012.

[14] A.-S. Puydupin-Jamin, M. Johnson, and T. Bretl. A convex approach to inverse optimal control and its application to modeling human locomotion. In *IEEE ICRA*, 2012.

[15] P. Englert, N. A. Vien, and M. Toussaint. Inverse kkt — learning cost functions of manipulation tasks from demonstrations. *IJRR*, 36(13-14):1474–1488, 2017.

[16] S. Schaal, A. Ijspeert, and A. Billard. Computational Approaches to Motor Learning by Imitation. *Philosophical Transactions of the Royal Society of London*, 358:537–547, 2003.

[17] A. Paraschos, C. Daniel, J. Peters, and G. Neumann. Probabilistic Movement Primitives. In *NIPS*, 2013.

[18] P. Pastor, M. Kalakrishnan, S. Chitta, E. Theodorou, and S. Schaal. Skill learning and task outcome prediction for manipulation. In *IEEE ICRA*, 2011.

[19] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 25(2), 2013.

[20] N. Jetchev and M. Toussaint. TRIC: Task space retrieval using inverse optimal control. *Autonomous Robots*, 37(2):169–189, 2014.

[21] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *ICML*, 2016.

[22] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[23] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998.

[24] L. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration for fast path planning. In *IEEE ICRA*, 1994.

[25] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, 30:846–894, 2011.

[26] Z. Kingston, M. Moll, and L. E. Kavraki. Sampling-based methods for motion planning with constraints. *Annual review of control, robotics, and autonomous systems*, 1:159–185, 2018.

[27] B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.

[28] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *IEEE ICRA*, 2018.

[29] R. Madaan, S. Zeng, B. Okorn, and S. Scherer. Learning adaptive sampling distributions for motion planning by self-imitation. *IEEE IROS Workshop*, 2018.

[30] M. Phillips, B. Cohen, S. Chitta, and M. Likhachev. E-graphs: Bootstrapping planning with experience graphs. *Robotics: Science and Systems*, 2012.

[31] I. Havoutis and S. Ramamoorthy. Motion synthesis through randomized exploration on submanifolds of configuration space. In *Robot Soccer World Cup*, pages 92–103. Springer, 2009.

[32] F. Zha, Y. Liu, W. Guo, P. Wang, M. Li, X. Wang, and J. Li. Learning the metric of task constraint manifolds for constrained motion planning. *Electronics*, 7(12):395, 2018.

[33] W. M. Boothby. *An introduction to differentiable manifolds and Riemannian geometry; 2nd ed.* Pure Appl. Math. Academic Press, Orlando, FL, 1986.

[34] P. Englert, I. M. R. Fernández, R. K. Ramachandran, and G. S. Sukhatme. Sampling-Based Motion Planning on Manifold Sequences. arXiv:2006.02027, 2020.

[35] N. Kambhatla and T. K. Leen. Dimension Reduction by Local Principal Component Analysis. *Neural Comput.*, 9(7):1493–1516, Oct. 1997.

[36] S. Deutsch and G. Medioni. Unsupervised learning using the tensor voting graph. In *Scale Space and Variational Methods in Computer Vision*, pages 282–293, 2015.

[37] T.-J. Chin and D. Suter. Out-of-sample extrapolation of learned manifolds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1547–1556, 2008.

[38] C. Bellinger, C. Drummond, and N. Japkowicz. Manifold-based synthetic oversampling with manifold conformance estimation. *Machine Learning*, 107(3):605–637, 2018.

[39] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IEEE IROS*, 2012.

[40] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *CoRR*, abs/1312.6114, 2014.

[41] K. Thopalli, R. Anirudh, J. J. Thiagarajan, and P. Turaga. Multiple subspace alignment improves domain adaptation. In *ICASSP 2019*, 2019.

[42] X. He. Quantum subspace alignment for domain adaptation. arXiv:2001.02472, 2020.

[43] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS Workshop*, 2017.

[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems, 2015.

Supplementary Materials

We provide the following supplementary material to enhance the main paper:

- **Orthogonal Subspace Alignment** – In Section A, we describe the details of the orthogonal subspace alignment and provide an algorithm that shows its step-by-step computations.
- **Additional Experiments** – In order to thoroughly evaluate the components of our approach, we present some additional experimental results in Section B.

A Orthogonal Subspace Alignment (OSA)

In previous work [41, 42], subspace alignment techniques – without orthogonality constraints – have been introduced to improve domain adaptation. For the purposes of this paper, we require a subspace alignment algorithm that preserves orthogonality of the subspaces being aligned, which we present in this section.

Given a set of orthonormal vectors $\mathcal{F} = \{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_d\}$ which spans a *space*, the matrix $\mathbf{B} = [\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \ldots \quad \boldsymbol{b}_d] \in \mathbb{R}^{d \times d}$ belongs to the Orthogonal Group $O(d)$. The Orthogonal Group has two connected components, where one connected component called the Special Orthogonal Group $SO(d)$ is characterized by determinant 1, and the other is characterized by determinant -1. However, if $\mathbf{B} = [\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \ldots \quad \boldsymbol{b}_d]$ has determinant 1 (i.e. if $\mathbf{B} \in SO(d)$), then substituting $\boldsymbol{b}_1$ with its additive inverse $(-\boldsymbol{b}_1)$ will result in $\bar{\mathbf{B}} = [-\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \ldots \quad \boldsymbol{b}_d]$ with determinant -1. Aligning two coordinate frames $\mathcal{F}^a$ and $\mathcal{F}^c$ to have a common origin and associated basis matrices $\mathbf{B}_a$ and $\mathbf{B}_c$, respectively, is equivalent to finding an $\mathbf{R} \in SO(d)$ such that $\mathbf{B}_a\mathbf{R} = \mathbf{B}_c$. The solution to this problem exists if and only if $\mathbf{B}_a$ and $\mathbf{B}_c$ come from the same connected component of $O(d)$, i.e. if either both $\mathbf{B}_a, \mathbf{B}_c \in SO(d)$ or both determinants of $\mathbf{B}_a$ and $\mathbf{B}_c$ are -1.

For a *subspace* such as the normal space $N_{\mathbf{q}}M$ associated with an on-manifold data point $\mathbf{q}$ on M spanned by the eigenvectors $\mathcal{F}_N = \{\boldsymbol{v}_{d-l+1}, \ldots, \boldsymbol{v}_d\}$, the concept of a determinant does not apply to $\mathbf{V}_{\mathrm{N}} = [\boldsymbol{v}_{d-l+1} \quad \ldots \quad \boldsymbol{v}_d] \in \mathbb{R}^{d \times l}$, as it is not a square matrix. However, the normal space $N_{\mathbf{q}}M$ can be described with infinitely-many orthonormal bases $\mathbf{V}_{\mathrm{N}0}$, $\mathbf{V}_{\mathrm{N}1}$, $\mathbf{V}_{\mathrm{N}2}$, ... $\mathbf{V}_{\mathrm{N}\infty}$ where the set of column vectors of each is an orthonormal basis of $N_{\mathbf{q}}M$. Each of these is a member of $\mathbb{R}^{d \times l}$. Moreover, we can pick the transpose of one of them, for example $\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}$, as a projection matrix, and $\mathbf{V}_{\mathrm{N}0}$ as the inverse projection matrix. Applying the projection operation to each of the orthonormal bases, we get $\mathbf{W}_{\mathrm{N}0} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{I}_{l \times l}$, $\mathbf{W}_{\mathrm{N}1} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}1}$, $\mathbf{W}_{\mathrm{N}2} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}2}$, ... $\mathbf{W}_{\mathrm{N}\infty} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}\infty}$, and we will show that $\mathbf{W}_{\mathrm{N}0}$, $\mathbf{W}_{\mathrm{N}1}$, $\mathbf{W}_{\mathrm{N}2}$, ..., $\mathbf{W}_{\mathrm{N}\infty}$ are members of $O(l)$, which also has two connected components like $O(d)$. To show this, first note that although $\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{I}_{l \times l}$, the matrix $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}} \neq \mathbf{I}_{d \times d}$. Hence, for any matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ in general, $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{A} \neq \mathbf{A}$. However, we will show that $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{v}$ for any vector $\boldsymbol{v}$ in the vector space $N_{\mathbf{q}}M$. Suppose $\mathbf{V}_{\mathrm{N}0} = [\boldsymbol{b}_1 \quad \boldsymbol{b}_2 \quad \ldots \quad \boldsymbol{b}_l] \in \mathbb{R}^{d \times l}$, then we can write $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}} = \sum_{i=1}^{l} \boldsymbol{b}_i\boldsymbol{b}_i^{\mathrm{T}}$. Since the collection $\{\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_l\}$ spans the vector space $N_{\mathbf{q}}M$, any vector $\boldsymbol{v}$ in this vector space can be expressed as $\boldsymbol{v} = \sum_{i=1}^{l} \alpha_i\boldsymbol{b}_i$. Moreover, $\boldsymbol{b}_i^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{b}_i^{\mathrm{T}} \sum_{j=1}^{l} \alpha_j\boldsymbol{b}_j = \alpha_i$ for any $i = 1, 2, ..., l$, because by definition of orthonormality $\boldsymbol{b}_i^{\mathrm{T}}\boldsymbol{b}_j = 1$ for $i = j$ and $\boldsymbol{b}_i^{\mathrm{T}}\boldsymbol{b}_j = 0$ for $i \neq j$. Hence, $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\boldsymbol{v} = (\sum_{i=1}^{l} \boldsymbol{b}_i\boldsymbol{b}_i^{\mathrm{T}})\boldsymbol{v} = \sum_{i=1}^{l}(\boldsymbol{b}_i^{\mathrm{T}}\boldsymbol{v})\boldsymbol{b}_i = \sum_{i=1}^{l} \alpha_i\boldsymbol{b}_i = \boldsymbol{v}$. Similarly, because the column vectors of $\mathbf{V}_{\mathrm{N}0}$, $\mathbf{V}_{\mathrm{N}1}$, $\mathbf{V}_{\mathrm{N}2}$, ..., $\mathbf{V}_{\mathrm{N}\infty}$ are all inside the vector space $N_{\mathbf{q}}M$, it follows that $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{V}_{\mathrm{N}0}$, $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}1} = \mathbf{V}_{\mathrm{N}1}$, $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}2} = \mathbf{V}_{\mathrm{N}2}$, ..., $\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}\infty} = \mathbf{V}_{\mathrm{N}\infty}$. Similarly, it can be shown that $\mathbf{V}_{\mathrm{N}i}\mathbf{V}_{\mathrm{N}i}^{\mathrm{T}}\boldsymbol{v} = \boldsymbol{v}$ for any vector $\boldsymbol{v}$ in the vector space $N_{\mathbf{q}}M$ for any $i = 0, 1, 2, ..., \infty$. Furthermore, $\mathbf{W}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{W}_{\mathrm{N}0} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}(\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0}) = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{I}_{l \times l}$, $\mathbf{W}_{\mathrm{N}1}^{\mathrm{T}}\mathbf{W}_{\mathrm{N}1} = \mathbf{V}_{\mathrm{N}1}^{\mathrm{T}}(\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}1}) = \mathbf{V}_{\mathrm{N}1}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}1} = \mathbf{I}_{l \times l}$, ... $\mathbf{W}_{\mathrm{N}\infty}^{\mathrm{T}}\mathbf{W}_{\mathrm{N}\infty} = \mathbf{V}_{\mathrm{N}\infty}^{\mathrm{T}}(\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}\infty}) = \mathbf{V}_{\mathrm{N}\infty}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}\infty} = \mathbf{I}_{l \times l}$, and $\mathbf{W}_{\mathrm{N}0}\mathbf{W}_{\mathrm{N}0}^{\mathrm{T}} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}(\mathbf{V}_{\mathrm{N}0}\mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0}) = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{I}_{l \times l}$, $\mathbf{W}_{\mathrm{N}1}\mathbf{W}_{\mathrm{N}1}^{\mathrm{T}} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}(\mathbf{V}_{\mathrm{N}1}\mathbf{V}_{\mathrm{N}1}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0}) = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{I}_{l \times l}$, ... $\mathbf{W}_{\mathrm{N}\infty}\mathbf{W}_{\mathrm{N}\infty}^{\mathrm{T}} = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}(\mathbf{V}_{\mathrm{N}\infty}\mathbf{V}_{\mathrm{N}\infty}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0}) = \mathbf{V}_{\mathrm{N}0}^{\mathrm{T}}\mathbf{V}_{\mathrm{N}0} = \mathbf{I}_{l \times l}$. All these show that $\mathbf{W}_{\mathrm{N}0}$, $\mathbf{W}_{\mathrm{N}1}$, $\mathbf{W}_{\mathrm{N}2}$, ..., $\mathbf{W}_{\mathrm{N}\infty} \in O(l)$. Moreover, using $\mathbf{V}_{\mathrm{N}0}$ as the inverse projection matrix, we get $\mathbf{V}_{\mathrm{N}0} = \mathbf{V}_{\mathrm{N}0}\mathbf{W}_{\mathrm{N}0}$, $\mathbf{V}_{\mathrm{N}1} = \mathbf{V}_{\mathrm{N}0}\mathbf{W}_{\mathrm{N}1}$, $\mathbf{V}_{\mathrm{N}2} = \mathbf{V}_{\mathrm{N}0}\mathbf{W}_{\mathrm{N}2}$, ... $\mathbf{V}_{\mathrm{N}\infty} = \mathbf{V}_{\mathrm{N}0}\mathbf{W}_{\mathrm{N}\infty}$. Therefore, there is a one-to-one mapping between $\mathbf{V}_{\mathrm{N}0}$, $\mathbf{V}_{\mathrm{N}1}$, $\mathbf{V}_{\mathrm{N}2}$, ..., $\mathbf{V}_{\mathrm{N}\infty}$ and $\mathbf{W}_{\mathrm{N}0}$, $\mathbf{W}_{\mathrm{N}1}$, $\mathbf{W}_{\mathrm{N}2}$, ..., $\mathbf{W}_{\mathrm{N}\infty}$. Furthermore, between any two of $\mathbf{V}_{\mathrm{N}0}$, $\mathbf{V}_{\mathrm{N}1}$, $\mathbf{V}_{\mathrm{N}2}$, ..., $\mathbf{V}_{\mathrm{N}\infty}$, e.g. $\mathbf{V}_{\mathrm{N}i}$ and $\mathbf{V}_{\mathrm{N}j}$, there exists $\mathbf{R}_N \in SO(l)$ such that $\mathbf{V}_{\mathrm{N}i}\mathbf{R}_N = \mathbf{V}_{\mathrm{N}j}$ if their $SO(l)$ projections $\mathbf{W}_{\mathrm{N}i}$ and $\mathbf{W}_{\mathrm{N}j}$ both are members of the same connected component of $O(l)$.

Now, suppose for nearby on-manifold data points $\mathbf{q}_a$ and $\mathbf{q}_c$, their approximate normal spaces $N_{\mathbf{q}_a}M$ and $N_{\mathbf{q}_c}M$ are spanned by eigenvector bases $\mathcal{F}_N^a = \{\boldsymbol{v}_{d-l+1}^a, \ldots, \boldsymbol{v}_d^a\}$ and $\mathcal{F}_N^c = \{\boldsymbol{v}_{d-l+1}^c, \ldots, \boldsymbol{v}_d^c\}$, respectively. Due to the curvature on the manifold M, the normal spaces $N_{\mathbf{q}_a}M$ and $N_{\mathbf{q}_c}M$ may intersect, but in general are different subspaces of $\mathbb{R}^{d\times d}$. For the purpose of aligning the basis of $N_{\mathbf{q}_a}M$ to the basis of $N_{\mathbf{q}_c}M$, one may think to do projection of the basis vectors of $N_{\mathbf{q}_a}M$ into $N_{\mathbf{q}_c}M$. Problematically, this projection may result in a non-orthogonal basis of $N_{\mathbf{q}_c}M$. Hence, we resort to an iterative method using a differentiable Special Orthogonal Group $SO(l)$. In particular, we form an $l \times l$ skew-symmetric matrix $\mathbf{L} \in so(l)$ with $l(l-1)/2$ differentiable parameters –where $so(l)$ is the Lie algebra of $SO(l)$, i.e. the set of all skew-symmetric $l \times l$ matrices–, and transform it through a differentiable exponential mapping (or matrix exponential) to get $\mathbf{R}_N = \exp(\mathbf{L})$ with $\exp : so(l) \rightarrow SO(l)$. With $\mathbf{V}_{\mathrm{N}}^a = \begin{bmatrix}\boldsymbol{v}_{d-l+1}^a & \cdots & \boldsymbol{v}_d^a\end{bmatrix}$ and $\mathbf{V}_{\mathrm{N}}^c = \begin{bmatrix}\boldsymbol{v}_{d-l+1}^c & \cdots & \boldsymbol{v}_d^c\end{bmatrix}$, we can do an iterative training process to minimize the alignment error between $\mathbf{V}_{\mathrm{N}}^a\mathbf{R}_N$ and $\mathbf{V}_{\mathrm{N}}^c$, that is $\mathcal{L}_{\text{osa}} = \left\|\mathbf{I}_{l\times l} - (\mathbf{V}_{\mathrm{N}}^a\mathbf{R}_N)^{\mathrm{T}}\mathbf{V}_{\mathrm{N}}^c\right\|_2^2$. Depending on whether both $\mathbf{W}_{\mathrm{N}}^a$ and $\mathbf{W}_{\mathrm{N}}^c$ (which are the projections of $\mathbf{V}_{\mathrm{N}}^a$ and $\mathbf{V}_{\mathrm{N}}^c$, respectively, to $O(l)$) are members of the same connected component of $O(l)$ or not, this alignment process may succeed or fail. However, if we define $\bar{\mathbf{V}}_{\mathrm{N}}^a = \begin{bmatrix}-\boldsymbol{v}_{d-l+1}^a & \boldsymbol{v}_{d-l+2}^a & \cdots & \boldsymbol{v}_d^a\end{bmatrix}$ and $\bar{\mathbf{V}}_{\mathrm{N}}^c = \begin{bmatrix}-\boldsymbol{v}_{d-l+1}^c & \boldsymbol{v}_{d-l+2}^c & \cdots & \boldsymbol{v}_d^c\end{bmatrix}$, two out of the four pairs $(\mathbf{V}_{\mathrm{N}}^a, \mathbf{V}_{\mathrm{N}}^c)$, $(\bar{\mathbf{V}}_{\mathrm{N}}^a, \mathbf{V}_{\mathrm{N}}^c)$, $(\mathbf{V}_{\mathrm{N}}^a, \bar{\mathbf{V}}_{\mathrm{N}}^c)$, and $(\bar{\mathbf{V}}_{\mathrm{N}}^a, \bar{\mathbf{V}}_{\mathrm{N}}^c)$ will be pairs in the same connected component. Thus, two of these pairs will achieve minimum alignment errors after training the differentiable Special Orthogonal Groups $SO(l)$ on these pairs, indicating successful alignment. These are the main insights for our *local* alignment of neighboring normal spaces of on-manifold data points.

For the *global* alignment of the normal spaces, we represent the on-manifold data points as a graph. Our Orthogonal Subspace Alignment (OSA) is outlined in Algorithm 1. We begin by constructing a sparse graph of nearest neighbor connections of each on-manifold data point, followed by the construction of this graph into an (un-directed) minimum spanning tree (MST), and eventually the conversion of the MST to a directed acyclic graph (DAG). This graph construction is detailed in lines 2 - 8 of Algorithm 1.

Each directed edge in the DAG represents a pair of on-manifold data points whose normal spaces are to be aligned locally. Our insights for the local alignment of neighboring normal spaces are implemented in lines 9 - 21 of Algorithm 1. In the actual implementation, these local alignment computations are done as a vectorized computation which is faster than doing it in a for-loop as presented in Algorithm 1; this for-loop presentation is made only for the sake of clarity. We initialize the $l(l-1)/2$ differentiable parameters of the $l \times l$ skew-symmetric matrix $\mathbf{L}$ with near zero random numbers, which essentially will map to a near identity matrix $\mathbf{I}_{l\times l}$ of $\mathbf{R}_N$ via the $\exp()$ mapping, as stated in line 14 of Algorithm 1[3]. This is reasonable because we assume that most of the neighboring normal spaces are already/close to being aligned initially. We optimize the alignment of the four pairs $(\mathbf{V}_{\mathrm{N}}^a, \mathbf{V}_{\mathrm{N}}^c)$, $(\bar{\mathbf{V}}_{\mathrm{N}}^a, \mathbf{V}_{\mathrm{N}}^c)$, $(\mathbf{V}_{\mathrm{N}}^a, \bar{\mathbf{V}}_{\mathrm{N}}^c)$, and $(\bar{\mathbf{V}}_{\mathrm{N}}^a, \bar{\mathbf{V}}_{\mathrm{N}}^c)$ in lines 16 - 19 of Algorithm 1.

Once the local alignments are done, the algorithm then traverses the DAG in breadth-first order, starting from the root r, where the orientation of the root is already chosen and committed to. During the breadth-first traversal of the DAG, three things are done: First, the orientation of each point is chosen based on the minimum alignment loss; second, the local alignment transforms are compounded/integrated along the path from root to the point; and finally, the (globally) aligned orthogonal basis of each point is computed and returned as the result of the algorithm. These steps are represented by lines 22 – 48 of Algorithm 1.

[3]Although most of our ECoMaNN implementation is done in PyTorch [43], the OSA algorithm is implemented in TensorFlow [44], because at the time of implementation of the OSA algorithm, PyTorch did not support the differentiable matrix exponential (i.e. the exponential mapping) computation yet while TensorFlow did.

Algorithm 1 Orthogonal Subspace Alignment (OSA)

```
1: function OSA($\{(\mathbf{q} \in \mathcal{C}_M$, orthogonal basis stacked as matrix $\mathbf{V}_{\mathrm{N}}$ associated with $N_{\mathbf{q}}M)\}$)
2:     # construct a sparse graph between each data point $\mathbf{q} \in \mathcal{C}_M$ with its $H$ nearest neighbors,
3:     # followed by minimum spanning tree and directed acyclic graph computations
4:     # to obtain directed edges $\mathcal{E}$; $H$ needs to be chosen to be a value as small as possible that
5:     # still results in all non-root points $\{\mathbf{q} \in \mathcal{C}_M \backslash \{\mathbf{q}_r\}\}$ being reachable from the root point $\mathbf{q}_r$:
6:     $\mathcal{G} \leftarrow$ computeNearestNeighborsSparseGraph($\{\mathbf{q} \in \mathcal{C}_M\}, H$)
7:     $\mathcal{T} \leftarrow$ computeMinimumSpanningTree($\mathcal{G}$)
8:     $\mathcal{E} \leftarrow$ computeDirectedAcyclicGraphEdgesByBreadthFirstTree($\mathcal{T}$)
9:     for each directed edge $\boldsymbol{e} = (\mathbf{q}_c, \mathbf{q}_a) \in \mathcal{E}$ do
10:        Obtain $\mathbf{V}_{\mathrm{N}}^{a} = \begin{bmatrix}\boldsymbol{v}_{d-l+1}^{a} & \cdots & \boldsymbol{v}_{d}^{a}\end{bmatrix} \in \mathbb{R}^{d \times l}$ associated with the source subspace $N_{\mathbf{q}_a}M$
11:        Obtain $\mathbf{V}_{\mathrm{N}}^{c} = \begin{bmatrix}\boldsymbol{v}_{d-l+1}^{c} & \cdots & \boldsymbol{v}_{d}^{c}\end{bmatrix} \in \mathbb{R}^{d \times l}$ associated with the target subspace $N_{\mathbf{q}_c}M$
12:        Define $\bar{\mathbf{V}}_{\mathrm{N}}^{a} = \begin{bmatrix}-\boldsymbol{v}_{d-l+1}^{a} & \boldsymbol{v}_{d-l+2}^{a} & \cdots & \boldsymbol{v}_{d}^{a}\end{bmatrix} \in \mathbb{R}^{d \times l}$
13:        Define $\bar{\mathbf{V}}_{\mathrm{N}}^{c} = \begin{bmatrix}-\boldsymbol{v}_{d-l+1}^{c} & \boldsymbol{v}_{d-l+2}^{c} & \cdots & \boldsymbol{v}_{d}^{c}\end{bmatrix} \in \mathbb{R}^{d \times l}$
14:        Define differentiable $SO(l)$ $\mathbf{R}_{N}^{\overrightarrow{a}\overrightarrow{c}}$, $\mathbf{R}_{N}^{\overrightarrow{a}\overleftarrow{c}}$, $\mathbf{R}_{N}^{\overleftarrow{a}\overrightarrow{c}}$, and $\mathbf{R}_{N}^{\overleftarrow{a}\overleftarrow{c}}$, initialized near identity
15:        # try optimizing the alignment of the 4 possible pairs:
16:        $(\mathbf{R}_{N}^{\overrightarrow{a}\overrightarrow{c}}, \mathcal{L}_{\overrightarrow{a}\overrightarrow{c}}) \leftarrow$ iterativelyMinimizeAlignmentError($\mathbf{V}_{\mathrm{N}}^{a}\mathbf{R}_{N}^{\overrightarrow{a}\overrightarrow{c}}, \mathbf{V}_{\mathrm{N}}^{c}$)
17:        $(\mathbf{R}_{N}^{\overrightarrow{a}\overleftarrow{c}}, \mathcal{L}_{\overrightarrow{a}\overleftarrow{c}}) \leftarrow$ iterativelyMinimizeAlignmentError($\mathbf{V}_{\mathrm{N}}^{a}\mathbf{R}_{N}^{\overrightarrow{a}\overleftarrow{c}}, \bar{\mathbf{V}}_{\mathrm{N}}^{c}$)
18:        $(\mathbf{R}_{N}^{\overleftarrow{a}\overrightarrow{c}}, \mathcal{L}_{\overleftarrow{a}\overrightarrow{c}}) \leftarrow$ iterativelyMinimizeAlignmentError($\bar{\mathbf{V}}_{\mathrm{N}}^{a}\mathbf{R}_{N}^{\overleftarrow{a}\overrightarrow{c}}, \mathbf{V}_{\mathrm{N}}^{c}$)
19:        $(\mathbf{R}_{N}^{\overleftarrow{a}\overleftarrow{c}}, \mathcal{L}_{\overleftarrow{a}\overleftarrow{c}}) \leftarrow$ iterativelyMinimizeAlignmentError($\bar{\mathbf{V}}_{\mathrm{N}}^{a}\mathbf{R}_{N}^{\overleftarrow{a}\overleftarrow{c}}, \bar{\mathbf{V}}_{\mathrm{N}}^{c}$)
20:        # record optimized local alignment rotation matrices and its associated loss w/ the edge:
21:        Associate $(\mathbf{R}_{N}^{\overrightarrow{a}\overrightarrow{c}}, \mathcal{L}_{\overrightarrow{a}\overrightarrow{c}})$, $(\mathbf{R}_{N}^{\overrightarrow{a}\overleftarrow{c}}, \mathcal{L}_{\overrightarrow{a}\overleftarrow{c}})$, $(\mathbf{R}_{N}^{\overleftarrow{a}\overrightarrow{c}}, \mathcal{L}_{\overleftarrow{a}\overrightarrow{c}})$, $(\mathbf{R}_{N}^{\overleftarrow{a}\overleftarrow{c}}, \mathcal{L}_{\overleftarrow{a}\overleftarrow{c}})$ with $\boldsymbol{e}$
22:    # commit on the orientation of the root point as un-flipped ($\overrightarrow{r}$) instead of flipped ($\overleftarrow{r}$):
23:    $ori(r) = \overrightarrow{r}$
24:    # define the compound/global alignment rotation matrix of the root as an identity matrix:
25:    $\mathbf{R}_{G}^{r} = \mathbf{I}_{l \times l}$
26:    # aligned orthogonal basis of the root is:
27:    $\mathbf{V}_{\mathrm{N}}^{\text{aligned},r} = \mathbf{V}_{\mathrm{N}}^{r}$
28:    # do breadth-first traversal from root to:
29:    # (1) select the orientation $ori()$ of each point based on the minimum alignment loss,
30:    # (2) compound/integrate the local alignment transforms $\mathbf{R}_G$ along the path to the point,
31:    # (3) and finally compute the aligned orthogonal basis $\mathbf{V}_{\mathrm{N}}^{aligned}$:
32:    $Q = Queue()$
33:    $Q.enqueue(childrenOfNodeInGraph(r, \mathcal{E}))$
34:    while $size(Q) > 0$ do
35:        $d = Q.dequeue()$
36:        $Q.enqueue(childrenOfNodeInGraph(d, \mathcal{E}))$
37:        $p = parentOfNodeInGraph(d, \mathcal{E})$
38:        # select the local alignment rotation matrix based on the minimum alignment loss
39:        # among the two possibilities:
40:        if $\mathcal{L}_{\overrightarrow{d}\, ori(p)} < \mathcal{L}_{\overleftarrow{d}\, ori(p)}$ then
41:            $ori(d) = \overrightarrow{d}$
42:            $\mathbf{R}_{G}^{d} = \mathbf{R}_{N}^{\overrightarrow{d}\, ori(p)}\mathbf{R}_{G}^{p}$
43:            $\mathbf{V}_{\mathrm{N}}^{\text{aligned},d} = \mathbf{V}_{\mathrm{N}}^{d}\mathbf{R}_{G}^{d}$
44:        else
45:            $ori(d) = \overleftarrow{d}$
46:            $\mathbf{R}_{G}^{d} = \mathbf{R}_{N}^{\overleftarrow{d}\, ori(p)}\mathbf{R}_{G}^{p}$
47:            $\mathbf{V}_{\mathrm{N}}^{\text{aligned},d} = \bar{\mathbf{V}}_{\mathrm{N}}^{d}\mathbf{R}_{G}^{d}$
48:    return $\{\mathbf{V}_{\mathrm{N}}^{aligned}$ associated with $N_{\mathbf{q}}M$ for each $\mathbf{q} \in \mathcal{C}_M\}$
```

B Additional Experiments

B.1 Learning ECoMaNN on noisy data

We also evaluate ECoMaNN learning on noisy data. We generate a noisy unit sphere dataset and a noisy 3D unit circle with additive Gaussian noise of zero mean and standard deviation 0.01. After we train ECoMaNN on these noisy sphere and 3D circle datasets, we evaluate the model and obtain (82.00 ± 12.83) % and (89.33 ± 11.61) %, respectively, as the $P_{\bar{h}_M}$ metric.[4] These are still quite high success rates and outperform the VAE without noise.

B.2 Relationship between the number of augmentation levels and the projection success rate

We also perform an experiment to study the relationship between the number of augmentation levels (the maximum value of the positive integer i in the off-manifold points $\check{\mathbf{q}} = \mathbf{q} + i\epsilon\boldsymbol{u}$) and the projection success rate. As we vary this parameter at 1, 2, 3, and 7 on the Sphere dataset, the projection success rates are (5.00 ± 2.83) %, (12.33 ± 9.03) %, (83.67 ± 19.01) %, and (97.33 ± 3.77) %, respectively, showing that the projection success rate improves as the number of augmentation levels are increased. Increasing this parameter too high, however, would eventually have two problems: First, we empirically found data augmentation to be a computationally expensive step in training, and second, it would be possible to run into an issue like augmenting a point on a sphere beyond the center of a sphere (as mentioned in section 4.2).

[4]The small positive scalar ϵ needs to be chosen sufficiently large as compared to the noise level, so that the data augmentation will not create inconsistent data w.r.t. the noise.