Roto-translated Local Coordinate Frames For Interacting Dynamical Systems

Miltiadis Kofinas
University of Amsterdam
m.kofinas@uva.nl

Naveen Shankar Nagaraja
BMW Group
Naveen-Shankar.Nagaraja@bmw.de

Eftratios Gavves
University of Amsterdam
egavves@uva.nl

Abstract

Modelling interactions is critical in learning complex dynamical systems, namely systems of interacting objects with highly non-linear and time-dependent behaviour. A large class of such systems can be formalized as *geometric graphs*, *i.e.*, graphs with nodes positioned in the Euclidean space given an *arbitrarily* chosen global coordinate system, for instance vehicles in a traffic scene. Notwithstanding the arbitrary global coordinate system, the governing dynamics of the respective dynamical systems are invariant to rotations and translations, also known as *Galilean invariance*. As ignoring these invariances leads to worse generalization, in this work we propose local coordinate frames per node-object to induce roto-translation invariance to the geometric graph of the interacting dynamical system. Further, the local coordinate frames allow for a natural definition of anisotropic filtering in graph neural networks. Experiments in traffic scenes, 3D motion capture, and colliding particles demonstrate that the proposed approach comfortably outperforms the recent state-of-the-art.

1 Introduction

Modelling interacting dynamical systems –systems of interacting objects with highly non-linear and time-dependent behaviour– with neural networks is attracting a significant amount of interest [27, 3, 19] for its potential to learn long-term behaviours directly from observations. A large class of these systems consists of objects in the physical space, for instance pedestrians in a traffic scene [31, 37] or colliding subatomic particles [5]. These systems can be formalized as *geometric graphs*, in which the nodes describe the physical coordinates of the objects among other features. Kipf et al. [27] introduced Neural Relational Inference (NRI) to learn geometric graph dynamical systems using the variational autoencoding framework [25, 38]. Following [27], dynamic NRI [19] advocated sequential latent variable models to encode time-transient behaviours. Both approaches and the majority of learning algorithms for dynamical systems assume an *arbitrary global coordinate system* to encode time-transient interactions and model complex behaviours. In this work, we posit that taking into account the relative nature of dynamics is key to accurate modelling of interacting dynamical systems.

Represented by geometric graphs, learning algorithms of dynamical systems subscribe themselves to the Newtonian space. The absolute notion of Newtonian space and mechanics, however, determines that there exist infinite inertial frames that connect with each other by a rotation and translation. Each of these inertial frames is equivalent in that they can all serve as global coordinate frames and, thus, an arbitrary choice is made. Notwithstanding this arbitrariness, the dynamics of the system are invariant to the choice of a global coordinate frame up to a rotation and translation, in what is also known as *Galilean invariance*. Put otherwise, geometric graphs of interacting dynamical systems often exhibit symmetries that if left to their own devices lead models to subpar learning.

Inspired by the notion of Galilean invariance, we focus on inducing roto-translation invariance in interacting dynamical systems and their geometric graphs to sustain the effects of underlying

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

pathological symmetries. Symmetries, invariances and equivariances have attracted an increased interest with learning algorithms in the late years [11, 12, 53, 54, 46, 17]. The reason is that with the increasing complexity of new tasks and data, exploiting the symmetries improves sample efficiency [17, 41] by requiring fewer data points and gradient updates. To date the majority of works on exploiting symmetries or data augmentations are with static data [11, 44, 54]. We argue and show that invariance in representations of dynamic data is just as important, if not more, as it is critical in accounting for the inevitable increased pattern complexity and non-stationarity.

We induce roto-translation invariant representations in graphs by local coordinate frames. Each local coordinate frame is centered at a node-object in the geometric graph and rotated to match its angular position –yaw, pitch, and roll. Since all intermediate operations are performed on the local coordinate frames, the graph neural network is roto-translation invariant and the final transformed output is roto-translation equivariant. We obtain equivariance to global roto-translations by an inverse rotation that transforms the predictions back to the global coordinates.

We make the following three contributions. First, we introduce canonicalized roto-translated local coordinate frames for interacting dynamical systems formalized in geometric graphs. Second, by operating solely on these coordinate frames, we enable roto-translation invariant edge prediction and roto-translation equivariant trajectory forecasting. Third, we present a novel methodology for natural anisotropic continuous filters based on relative linear and angular positions of neighboring objects in the canonicalized local coordinate frames. We continue with a brief introduction of relevant background and then the description of our method. We present related work and finish with experiments and ablation studies on a number of settings, including modelling pedestrians in 2D traffic scenes, 3D particles colliding, and 3D human motion capture systems.

2 Background

2.1 Interacting dynamical systems and geometric graphs

An interacting dynamical system consists of $i = 1, \dots, N_t$ objects, whose position $\mathbf{p} = (p_x, p_y, p_z)^\top$ and velocity $\mathbf{u} = \frac{d\mathbf{p}}{dt} = (u_x, u_y, u_z)^\top$ in the Euclidean space are recorded over time t. The state of the i-th object at timestep t is described by $\mathbf{x}_i^t = [\mathbf{p}_i^t, \mathbf{u}_i^t]$, adopting for clarity a column vector notation and using $[\cdot, \cdot]$ to denote vector concatenation.

Over the past few years, a natural way that has emerged for organizing interacting dynamical systems is by geometric graphs [3, 27, 19] through space and time, $\mathcal{G} = \{\mathcal{G}^t\}_{t=1}^T$, where $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$ is the snapshot of the graph at timestep t. The nodes $\mathcal{V}^t = \left\{v_1^t, \dots, v_{N_t}^t\right\}$ of the graph correspond to the objects in the dynamical system, with v_i^t corresponding to the state of the i-th object, $\mathbf{x}_i^t$. The edges $\mathcal{E}^t \subseteq \left\{e_{j,i}^t = \left(v_j^t, v_i^t\right) \mid \left(v_j^t, v_i^t\right) \in \mathcal{V}^t \times \mathcal{V}^t\right\}$ of the graph, correspond to the interactions from node-object j to node-object i. We use $\mathcal{N}(i)$ to denote the graph neighbours of node v_i. In the absence of domain knowledge about how objects connect, for instance the links between atoms in molecules, the graph is fully connected. Explicit inference of the graph structure can be achieved by using latent edges $\mathbf{z}_{j,i}^t$ corresponding to the edges $e_{j,i}^t$.

Graph neural networks [42, 32, 18] exchange messages between neighbors and update the vertex and edge embeddings per layer, commonly referred to as message passing

$$\mathbf{h}_{j,i}^{(l)} = f_e^{(l)}\left(\left[\mathbf{h}_i^{(l-1)}, \mathbf{h}_{j,i}^{(l-1)}, \mathbf{h}_j^{(l-1)}\right]\right) \tag{1}$$

$$\mathbf{h}_i^{(l)} = f_v^{(l)}\left(\mathbf{h}_i^{(l-1)}, \square_{j \in \mathcal{N}(i)} \mathbf{h}_{j,i}^{(l)}\right), \tag{2}$$

where $\mathbf{h}_i^{(l)}$ is the embedding of node v_i at layer l and $\mathbf{h}_{j,i}^{(l)}$ is the embedding of edge $e_{j,i}$ at layer l. f_e, f_v denote differentiable functions such as MLPs and $\square$ denotes a permutation invariant function, commonly a summation or an average. Many graph neural networks rely on isotropic filters [26], although various ways [47, 36] to circumvent this constraint have also been explored.

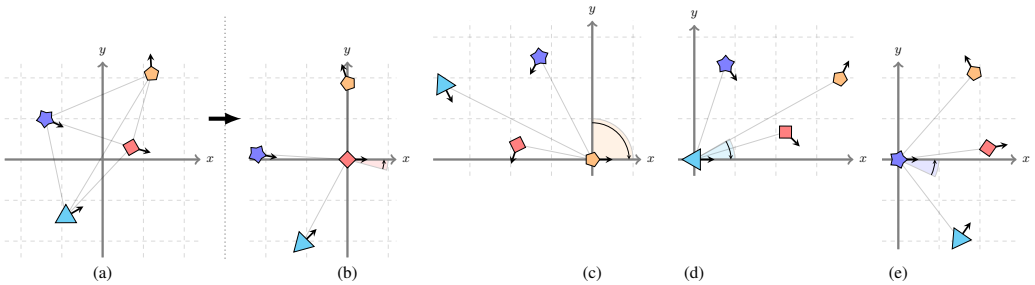


Figure 1: In (a), objects positioned in an arbitrary global 2D coordinate frame; arrows represent orientations. In (b)-(e), objects in the canonicalized local coordinate frames, translated to match the target object's position and rotated to match its orientation

2.2 Neural relational inference

Kipf et al. [27] proposed neural relational inference (NRI), a variational autoencoding inference model [25, 38] that explicitly infers the graph structure over a discrete latent graph and simultaneously learns the dynamical system. Given the input trajectories the encoder learns to infer interactions as latent edges $\mathbf{z}_{j,i} \in [0,1]^K$, sampled from a concrete distribution [34, 23] with K edge types. The decoder receives the inferred interactions and the past trajectories, and learns the dynamical system, $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z})$. The encoder is a regular graph neural network without explicitly taking time into account and learns to infer latent interactions by maximizing the evidence lower bound for the next time step. The decoder is another graph neural network that either assumes Markovian dynamics $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}) = p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{t}, \mathbf{z})$ or is recurrent through time. As the prior and encoder in NRI assume static interactions (*e.g.* whether forces between charged particles are attractive or repulsive), dynamic NRI [19] replaces them with a sequential relation prior based on past states, $p_\phi(\mathbf{z}|\mathbf{x}) = \prod_{t=1}^{T} p_\phi\big(\mathbf{z}^t|\mathbf{x}^{1:t}, \mathbf{z}^{1:t-1}\big)$, and an approximate relation posterior based on both the past and future states. The decoder is also reformulated as $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t})$, taking into account the dynamic nature of interactions.

2.3 Invariance and equivariance

Last, we give a very brief introduction to invariance and equivariance. A function $f : \mathcal{X} \to \mathcal{Y}$ is equivariant [54] under a group of transformations if every transformation $\pi \in \Pi$ of the input $\mathbf{x} \in \mathcal{X}$ can be associated with a transformation $\psi \in \Psi$ of the output, $\psi[f(\mathbf{x})] = f(\pi[\mathbf{x}])$. A special case is invariance, where $\Psi = \{\mathbb{I}\}$, the identity transformation, $f(\mathbf{x}) = f(\pi[\mathbf{x}])$.

In this work, we are interested in translation, *i.e.*, $f(\mathbf{x}) + \boldsymbol{\tau} = f(\mathbf{x} + \boldsymbol{\tau})$ with the translation vector $\boldsymbol{\tau}$, and rotation invariance/equivariance, *i.e.*, $\mathbf{Q}f(\mathbf{x}) = f(\mathbf{Q}\mathbf{x})$ using the rotation matrix $\mathbf{Q}$.

3 Roto-translation invariance with local coordinate frames

In this section we present our method, termed LoCS (**Lo**cal **C**oordinate frame**S**). We start with the derivation of roto-translated local coordinate frames and continue with the formulation of graph networks and continuous anisotropic filters operating in these frames.

3.1 Local coordinate frames

Starting from the spatio-temporal graph $\mathcal{G}$, we focus for clarity on pairs of node-objects in the same time step, $\mathbf{x}_i^t, \mathbf{x}_j^t$. In the real world, objects are not point particles and have a spatial extension. Central to our method is the use of the angular positions $\boldsymbol{\omega} = (\theta, \phi, \psi)^\top$, otherwise known as yaw, pitch and roll, that describe the orientation of a rigid body with respect to the axes of the coordinate system. We, thus, augment the states $\mathbf{x}_i^t$ with the angular positions, using $\mathbf{v}_i^t = [\mathbf{p}_i^t, \boldsymbol{\omega}_i^t, \mathbf{u}_i^t]$ to denote the augmented state that captures the angular position as well as the linear position and velocity.

Our method draws inspiration from Galilean invariance and inertial frames of reference that capture the relative locations and motions of all the objects in a system. We introduce N_t local coordinate frames, one per object in the system. For the i-th object, a local coordinate frame is one in which both the linear and the angular position lie on the origin. By the adoption of the local coordinates with respect to object-centric frames, the behaviors between objects will not depend on the arbitrary positions of objects in the absolute Newtonian space. In other words, the local coordinates offer invariance to global translations and rotations and do not bias the learning algorithm. Our goal, thereby, is to compute the relative local coordinates of all objects $j = 1, \dots, N_t$, while iterating over the i-th reference object.

The transformation from global to local coordinate systems is in two steps. Per target node, we first translate the origin to match its linear position by a translation transformation. Since velocities and angular positions are translation invariant, we only need to perform the translation to the linear positions. This gives us the relative positions $\mathbf{r}^t_{j,i} = \left[\mathbf{p}^t_j - \mathbf{p}^t_i\right]$. Then, we canonicalize the local coordinate frame to match the target object's orientation by a rotation transformation, described by the rotation matrix $\mathbf{Q}(\boldsymbol{\omega}_i)$. The coordinates of all other objects are analogously transformed given the i-th local coordinate frame. The rotations are performed independently and equivalently to the state components of the j-th object, namely the relative –due to the translation transformation performed first– linear positions, the angular positions and the velocities. A schematic overview of the proposed transformation in a 2D setting is presented in fig. 1. Using tensor operations, we compactly write down the transformed state as:

$$\tilde{\mathbf{R}}(\boldsymbol{\omega}) = \mathbf{Q}(\boldsymbol{\omega}) \oplus \mathbf{Q}(\boldsymbol{\omega}) \oplus \mathbf{Q}(\boldsymbol{\omega}) \tag{3}$$

$$\mathbf{v}^t_{j|i} = \tilde{\mathbf{R}}^{t\top}_i \left[\mathbf{r}^t_{j,i}, \boldsymbol{\omega}^t_j, \mathbf{u}^t_j\right] \tag{4}$$

where $\oplus$ denotes the direct sum operator that concatenates two matrices along the diagonal resulting in a block diagonal matrix and $\tilde{\mathbf{R}}^{t\top}_i = \tilde{\mathbf{R}}^{\top}(\boldsymbol{\omega}^t_i)$ to reduce notation clutter. The rotation matrix in eq. (3) has three entries. Each $\mathbf{Q}(\boldsymbol{\omega})$ independently transforms the relative linear positions, the angular positions and the velocities.

Local coordinate frames in 2D We start with the simpler case where the dynamical system resides in the 2D Euclidean space, for instance pedestrians in a traffic scene. In 2 dimensions, the angular position is a scalar value, namely the yaw angle θ. Thus, the rotation matrix for a target node v_i is:

$$\mathbf{Q}(\boldsymbol{\omega}_i) = \mathbf{Q}(\theta_i) = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} \tag{5}$$

Local coordinate frames in 3D When in the 3D Newtonian space the chain of transformations is the same; first, we translate the origin to match each target node's linear position, and then we rotate the local coordinate frames to match each object's orientation. The translation transformation is identical to the 2D case. However, the rotation transformation is more involved. For the 3D case, we must decompose $\mathbf{Q}(\boldsymbol{\omega})$ into 3 chained elemental rotations, described by the matrices $\mathbf{Q}_z(\theta)$, $\mathbf{Q}_y(\phi)$ and $\mathbf{Q}_x(\psi)$. $\mathbf{Q}_z(\theta)$ describes a rotation around the z-axis by an angle θ, $\mathbf{Q}_y(\phi)$ describes a rotation around the y-axis by an angle ϕ and $\mathbf{Q}_x(\psi)$ describes a rotation around the x-axis by an angle ψ. We use the following convention that dictates the order of rotations, $\mathbf{Q}(\boldsymbol{\omega}) = \mathbf{Q}_z(\theta)\mathbf{Q}_y(\phi)\mathbf{Q}_x(\psi)$. Each elemental rotation matrix has similar structure to eq. (5). We provide the complete description of all rotation matrices in appendix A.2. After the computation of the rotation matrix, the states are transformed identically to eq. (4).

The local coordinate frames are invariant with respect to global translations and rotations, either in 2 or 3 dimensions. We provide the detailed derivations of $\mathbf{Q}(\boldsymbol{\omega})$ for the 2D and 3D case in appendices A.1 and A.2, respectively, and a detailed proof in appendix A.3.

3.2 Approximate angular positions

In practice, we do not always have perfect information about the object states, such as the angular positions. In this case, we can approximate them using the angles of the velocity vector as a proxy. Specifically, in 2 dimensions, the angular position is a scalar value and is approximated using the azimuth angle of the polar representation of the velocity vector, $\theta = \tan^{-1}(u_y/u_x)$. In 3 dimensions, we transform velocities to spherical coordinates $(u_\rho, u_\theta, u_\phi)$ and use these angles to rotate the local

coordinate frame and approximate 2 out of the 3 angles. The angle θ is computed as above and $\phi = \cos^{-1}(u_z/\|\mathbf{u}\|_2)$. In this case, we retain invariance for the coordinates for which we have perfect knowledge, while we will have an invariance leakage for the approximate ones. That said, experiments show there is little to no consequences and accurate predictions are still attained.

3.3 Local coordinate frame graph neural networks

Having canonicalized the object states in the interacting system, we obtain representations that are invariant to global translations and rotations. Following [19], we formulate the core of the network as a variational autoencoder [25, 38] with latent edge types that change dynamically over time. The network receives the canonicalized representations as input and operates solely on the local coordinate systems. We infer the graph structure over a discrete latent graph and simultaneously learn the dynamical system. Learning the graph structure is a roto-translation invariant task; we want to predict the same edge distribution for each pair of vertices regardless of the global rotation of translation. In contrast, trajectory forecasting is a roto-translation equivariant task; a global translation and rotation to the input trajectories should affect the output trajectories equivalently. Following [19], we maximize the evidence lower bound, $\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}|\mathbf{x})]$. We provide the exact form of the loss components in appendix B.1.

Encoder and prior Our encoder and prior closely follow [19], described in section 2.2. First, we compute the local coordinate frame representations $\mathbf{v}^t_{j|i}$ per pair j, i (including self-loops) and per timestep t according to eq. (4). We then perform a number of message passing steps and obtain a feature vector per object pair. Omitting time indices for clarity, we have

$$\mathbf{h}^{(1)}_{j,i} = f^{(1)}_e\big([\mathbf{v}_{j|i}, \mathbf{v}_{i|i}]\big) \tag{6}$$

$$\mathbf{h}^{(1)}_i = f^{(1)}_v\left(g^{(1)}_v\big(\mathbf{v}_{i|i}\big) + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\mathbf{h}^{(1)}_{j,i}\right) \tag{7}$$

$$\mathbf{h}^{(2)}_{j,i} = f^{(2)}_e\Big(\Big[\mathbf{h}^{(1)}_i, \mathbf{h}^{(1)}_{j,i}, \mathbf{h}^{(1)}_j\Big]\Big) \tag{8}$$

The functions $f^{(1)}_e, f^{(1)}_v, f^{(2)}_e$ are MLPs and $g^{(1)}_v$ is a linear layer. We feed the embeddings $\mathbf{h}^{(2)}_{j,i}$ into 2 LSTMs [22]: one forward in time that computes the prior and one backwards in time for the encoder. The hidden state from the forward LSTM is used to compute the prior distribution, while the hidden states from both the forward and the backward LSTM are concatenated to compute the encoder distribution. The formulation is identical to [19]; the exact details can be found in appendix B.1.

During training, we sample interactions $\mathbf{z}^t_{j,i}$ from $q_\phi\big(\mathbf{z}^t_{j,i}|\mathbf{x}\big)$ using Gumbel-Softmax [34, 23]. We perform teacher-forcing during the whole training, and task the model to predict the trajectories only for one step ahead. During inference, we sample interactions from the prior distribution.

Decoder As mentioned in section 2.2, the decoder $p_\theta(\mathbf{x}|\mathbf{z}) = \prod_{t=1}^{T} p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t})$ is tasked with predicting future trajectories given past and present trajectories as well as the predicted relations. As proposed by [19, 27] we can have either a Markovian or a recurrent decoder, depending on the governing dynamics. In many settings, like colliding elementary particles in physics, the governing dynamics satisfy the Markov property, $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t}) = p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, \mathbf{z}^t)$. In this case, the decoder is implemented with a graph neural network similar to [19]. In many real-world applications, however, the Markovian assumption does not hold. In that case, the graph neural network also features a GRU unit that learns a recurrent hidden state during the message passing.

We can use local coordinates frames with both types of decoders, as defined in [19, 27], with the difference that the message passing is performed with the local coordinate frame representations $\mathbf{v}^t_{j|i}$, so that we attain roto-translation invariance,

$$\mathbf{m}^t_{j,i} = \sum_k z^t_{(j,i),k} f^k\Big(\Big[\mathbf{v}^t_{j|i}, \mathbf{v}^t_{i|i}\Big]\Big) \tag{9}$$

$$\mathbf{m}^t_i = f^{(3)}_v\left(g^{(3)}_v\Big(\mathbf{v}^t_{i|i}\Big) + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\mathbf{m}^t_{j,i}\right). \tag{10}$$

The output of the model per time step comprises position and velocity predictions for the next time step. As common in the literature [19], we predict the difference in position and velocity from the previous time step, which equals the velocity and acceleration respectively, and numerically integrate to make predictions. While computations up to the output layer are roto-translation invariant, the predictions must be roto-translation equivariant, so that global roto-translations to the inputs affect the outputs equivalently. To transform predictions back to the global coordinate frame and achieve roto-translation equivariance, we do an inverse rotation by $\mathbf{R}(\boldsymbol{\omega}_i^t) = \mathbf{Q}(\boldsymbol{\omega}_i^t) \oplus \mathbf{Q}(\boldsymbol{\omega}_i^t)$, and then integrate numerically, *i.e.*, $\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{R}(\boldsymbol{\omega}_i^t) \cdot \boldsymbol{\Delta}\mathbf{x}_i^{t+1}$. We provide the definitions for both decoders in appendix B.1, and a detailed proof on equivariance to global roto-translations in appendix A.3.

3.4 Anisotropic filtering

One of the main reasons why filters in graph neural networks are isotropic is the inherent absence of an invariant coordinate frame. In a geometric graph dynamical system, object positions can serve this role. We, thus, use the local roto-translation invariant coordinate frames for anisotropic filtering, using weights that depend on the relative linear positions and angular positions of objects given the central i-th object. Similar to [45], our filter generating network, implemented by an MLP, is a matrix field that maps relative positions and orientation tuples to graph network filters, *i.e.* weight matrices, $\mathbf{W}_{\mathcal{F}} : \mathbb{R}^D \times \mathcal{S}^{|\boldsymbol{\omega}|} \to \mathbb{R}^{D_{\text{out}} \times D_{\text{in}}}$. The anisotropic filters replace the isotropic ones in updating the latent edge representations, weighing neighbors according to their positions, $\mathbf{h}_{j,i}^t = \mathbf{W}_{\mathcal{F}}\big(\big[\mathbf{Q}^\top(\boldsymbol{\omega}_i^t) \cdot \mathbf{r}_{j,i}^t, \mathbf{Q}^\top(\boldsymbol{\omega}_i^t) \cdot \boldsymbol{\omega}_j^t\big]\big) \cdot \Big[\mathbf{v}_{j|i}^t, \mathbf{v}_{i|i}^t\Big]$.

3.5 Data normalization

While the graph neural networks are roto-translation invariant and equivariant in the intermediate and the output layers respectively, we must also make sure that the pre- and post-processing of data are appropriate. The common practices of min-max normalization or z-score normalization are unsuitable because they anisotropically scale and translate the input and output position and velocities. That is, these transformations change the directions of the velocity vectors non-equivalently. This is counter-intuitive, since velocities are not treated as geometric entities but as generic additional dimensions to the features. For instance, as translation equals a vector subtraction changing the magnitude and the direction of vectors, translating the velocities removes any notion of speed from the input to the neural network. What is more, the scaling operations apply anisotropic transformations, affecting each axis differently.

We instead opt for a much simpler data normalization scheme that is more geometrically oriented and suitable for roto-translation invariance with local coordinate frames. This scheme does not perform any translation operations, since local coordinate frames naturally tend to center data around the origin; besides, they are invariant to a mere isotropic translation to the node positions. For the scaling operation, we opt for a simple isotropic transformation that shrinks relative positions and velocities equivalently across all axes. We scale the inputs, both positions and velocities, by the maximum speed (velocity norm) in the training set, $s_{\text{max}} = \max_i \|\mathbf{u}_i\|$, that is $\mathbf{x}' = \mathbf{S}^{-1}\mathbf{x}$, where $\mathbf{S} = \text{diag}(s_{\text{max}} \cdot \mathbf{1})$. During post-processing, we can convert our predictions to actual units, *e.g.* m and m/s, by applying the inverse transformation, $\mathbf{x} = \mathbf{S}\mathbf{x}'$. We term this operation *speed normalization*.

4 Related work

Learning dynamical systems & trajectory forecasting In the late years, and alongside NRI [27] and dNRI [19], many have studied learning dynamical systems [3, 40]. Further, many works have focused on the problems of pedestrian motion prediction and traffic scene trajectory forecasting [1, 28, 21, 35, 39]. A number of works [1, 39] uses distance-based heuristics to create the graph adjacency and estimate interactions. Kosaraju et al. [28] use self-attention to predict the influence of neighbouring nodes. Both approaches are different from our work, since we explicitly predict the latent graph structure and perform inference on it.

Equivariant deep learning Equivariant neural networks [11, 12, 53, 54, 46] have risen in popularity over the past few years, demonstrating high effectiveness and parameter efficiency. Schütt et al. [44] use radial basis functions on pair-wise node distances to generate continuous filters and perform mes-

sage passing using depth-wise separable convolutions. Fuchs et al. [17] introduce SE(3)-transformers by incorporating spherical harmonics in a transformer network, resulting in a 3D roto-translation equivariant attention network. de Haan et al. [13] propose anisotropic gauge equivariant kernels for graph networks on meshes based on neighbouring vertex angles and parallel transport. Walters et al. [50] propose rotationally equivariant continuous convolutions for 2D trajectory prediction. Closer to our work is the work of Satorras et al. [41]. They propose a graph network that leverages the rotation equivariant relative position and roto-translation invariant euclidean distance between node pairs in a novel message passing scheme that updates node features as well as node coordinate embeddings. Different from our work, they do not capitalize on orientations and velocities of neighbouring objects. Our work leverages these impactful quantities expressed in local coordinate frames to make more reliable predictions.

Anisotropic filtering Graph neural networks [42, 32, 18] have been used extensively for modeling dynamical systems [55, 35, 27, 19, 3]. Several works [35] incorporate isotropic graph filters [26, 20] as part of the graph network. However, these isotropic filters have a global weight sharing scheme, *i.e.* they use a single weight matrix for all neighbours, which amounts to a linear transformation over the aggregated neighbour information. Velickovic et al. [47] use self-attention [2] and [36] use Gaussian Mixture Models (GMMs) based on relative neighbour positions.

Other works address this issue by proposing continuous filters on graphs and point clouds [24, 51, 45, 50, 44, 16]. Highly related to our work is the work of Simonovsky and Komodakis [45]. They generalize convolution to arbitrary graphs and introduce dynamically generated filters based on edge attributes to perform continuous anisotropic convolutions on graph signals and point clouds. Differently, though, they do not operate under roto-translated local coordinate systems. This results in diminished parameter efficiency and weight sharing that they compensate for with data augmentation.

5 Experiments

We evaluate the proposed method, LoCS, on 2D and 3D geometric graph dynamical systems from the literature. In 2D, we evaluate on a synthetic physics simulation dataset proposed by dNRI [19] and on traffic trajectory forecasting [4]. In 3D, we evaluate on an 3D-extended version of the charged particles [27] and on a motion capture dataset [10]. We compare with NRI [27], dNRI [19], and the very recent EGNN [41]. For all methods we use publicly available code from [19, 41]. For EGNN, we autoregressively feed the output as the input to the next timestep. The full implementations details are in appendix B.3. Our code, data, and models will be available online[1].

Our architecture closely follows dNRI[26, 19]. Unless otherwise specified, all common layers have the same structure, and we use the same number of latent edge types. Following [26, 19], we report the mean squared error of positions and velocities over time. We compute errors in the original unnormalized data space for a fair comparison across different data normalization techniques and scales, $E(t) = \frac{1}{ND}\sum_{n=1}^{N} \|\mathbf{x}_n^t - \hat{\mathbf{x}}_n^t\|_2^2$. We also report the L_2 norm errors for positions (displacement errors), $E_p(t) = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{p}_n^t - \hat{\mathbf{p}}_n^t\|_2$, and velocities, $E_u(t) = \frac{1}{N}\sum_{n=1}^{N}\|\mathbf{u}_n^t - \hat{\mathbf{u}}_n^t\|_2$, separately to gain insights. L_2 norm errors are in the same scale with the respective variables (position and velocity) and, thus, more interpretable than MSE. We always ran experiments using 5 different random initialization seeds and report the mean and the standard deviation. We plot each method with a different color, as well as a different marker for color-blind friendly visualizations. The x-coordinates of the markers are chosen for aesthetic purposes; they differ across settings and they are not meant to convey extra information. Here we report the main results and visualizations and provide much more extensive examples in the appendix.

5.1 Synthetic dataset

On the synthetic 2D physics simulation we use the same experimental settings as in [19]. The dataset comprises scenes with three particles. Two of the particles move with a constant, randomly initialized velocity and the third particle is initialized with random velocity but pushed away when close to one of the others. Scenes last for 50 timesteps. For evaluation, we use the first 25 timesteps as input and the models are tasked with predicting the following

[1] `https://github.com/mkofinas/locs`

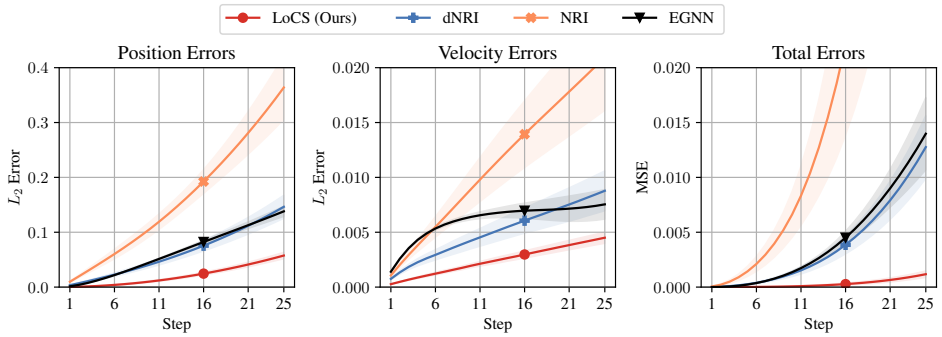
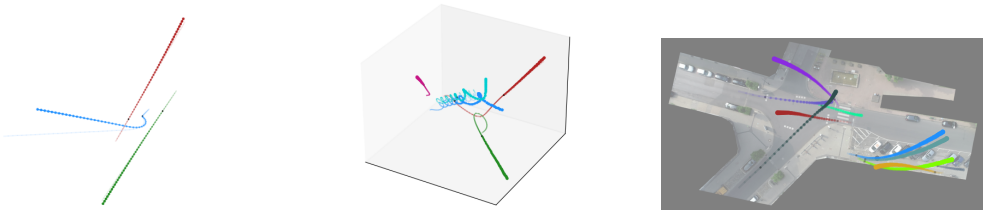


Figure 2: Results on synthetic dataset



(a) (b) (c)

Figure 3: LoCS predictions on (a) synthetic dataset, (b) charged particles and (c) inD.

25 timesteps. We report the results in fig. 2 and plot qualitative examples fig. 3a, more in fig. 7 in appendix C.1. We also report the average F1 score for relation prediction in table 1. In all visualizations, each color denotes a different object. Semi-transparent trajectories indicate the groundtruth future trajectories. Present timesteps are denoted by small black circles. Markers denote the timesteps, increasing in size as trajectories evolve through time. We observe that our method can reliably model interactions and outperforms competing methods in predicting future trajectories.

Table 1: Relation prediction F1 score on synthetic dataset

Method	NRI	dNRI	LoCS
F1	26.5	60.8	88.9

5.2 Charged particles

In the charged particles datasets the particles interact with one another via electrostatic forces. We extend the dataset by [27] from 2 to 3 dimensions and, following a similar approach to [17, 41], we remove the virtual boxes that confine the particle trajectories. We generate 30,000 scenes for training, 5,000 for validation and 5,000 for testing. Each scene comprises trajectories of 5 particles that carry either a positive or a negative charge. The forces are either attractive or repulsive according to physical laws. Following [27], training and validation scenes last for 49 timesteps. For evaluation, test scenes last for 20 additional timesteps (50 for visualization). We plot the MSE in fig. 4a and L_2 errors in Figure 11 in appendix D. LoCS has consistently lower errors, in total as well as individually for positions and velocities. We, further, provide qualitative results for 50 future time steps in fig. 3b, see more in fig. 8 in appendix C.2.

5.3 Traffic trajectory forecasting

The inD dataset [4] is a real-world 2D traffic trajectory forecasting dataset of pedestrians, vehicles, and cyclists, recorded at 4 different German traffic intersections. Traffic scenes contain a varying number of participants also changing over time. We follow the exact same setting as in dNRI. The dataset contains 36 recordings; we split them in 19/7/10 for training, validation and testing. We divide each scene into 50-step sequences. We use the first 5 timesteps as input and the model has to predict the remaining 45 time steps. We compare with dNRI, EGNN, and a GRU [7] baseline, but not with NRI since there are varying number of nodes. We plot the MSE in fig. 4b and L_2 errors in Figure 12 in appendix D and as before, LoCS outperforms competing methods consistently.

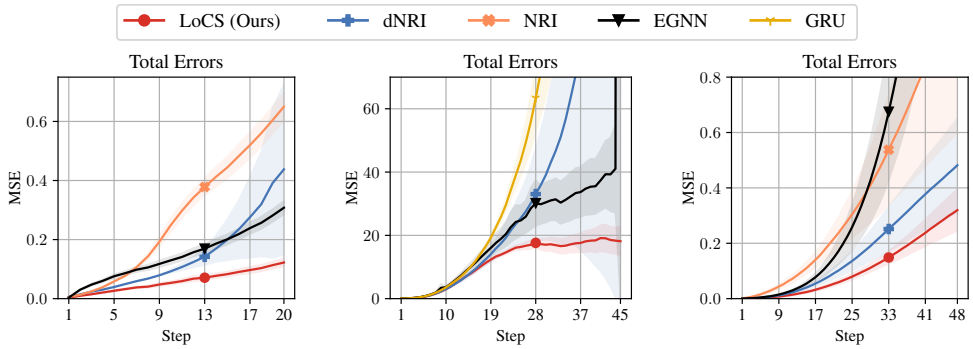


Figure 4: Total error curves in: (a) charged particles, (b) inD, (c) motion #35

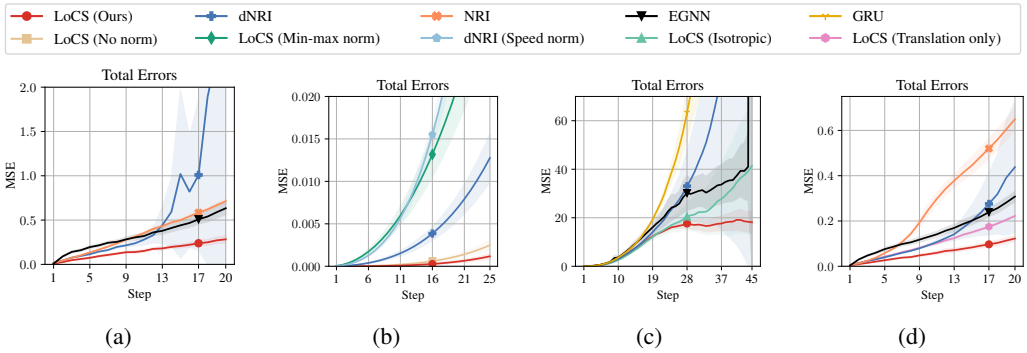


Figure 5: Total error curves in ablation experiments: (a) on highly interactive charged particles, (b) on the impact of speed normalization, (c) on the impact of isotropic filters, (d) on the impact of rotation.

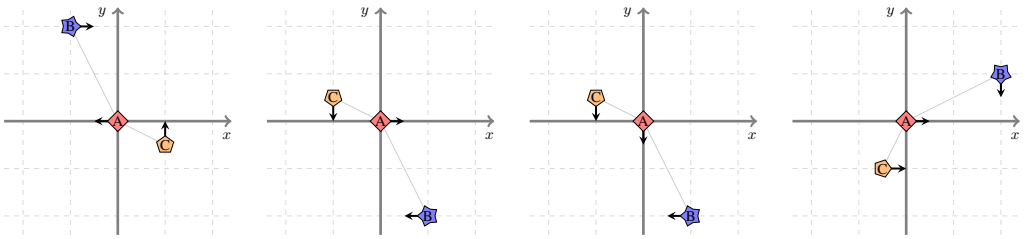
5.4 Motion capture

Last, we experiment with the CMU motion capture database [10] with 3D data, following the exact same setting as [27, 19] and studying the motion of subject #35. We train the models using sequences of 50 timesteps as input and evaluate on sequences of 99 time steps. We plot the MSE in fig. 4c and L_2 errors in fig. 13 in appendix D and observe that LoCS is attaining consistently lower errors.

5.5 Ablation experiments

High-intensity interactions We assess LoCS in fig. 5a in highly interactive scenarios by creating a subset of the charged particles test set (819 scenes – 16.38% of the original test set) in which a simple constant velocity model performs poorly (L_2 error > 1.5). With more and stronger interactions the proposed local coordinate frames performs even better relatively to competitors.

Speed normalization impact We assess the impact of speed normalization on the synthetic dataset in fig. 5b. We evaluate LoCS with and without speed normalization, as well as dNRI with speed normalization instead of min-max normalization. Results are shown in fig. 5b. We observe that when the inputs are normalized using min-max normalization, LoCS underperforms. Without any normalization, LoCS already performs better than other baselines. Using speed normalization improves the performance of LoCS even further. Finally, speed normalization is not the main cause for the improvements for LoCS. If it were, it should also benefit dNRI. We conclude that speed normalization is important for local coordinate frames to make sure equivariance is maintained after un-normalization. LoCS attains the highest accuracies when combined with speed normalization, although it works quite well even without it, while methods that are not equivariant like dNRI do not benefit from it.



(a) (b) (c) (d)

Figure 6: In a, b, c, translated-only local coordinate frames for object A in 3 different dynamical systems #1-#3. In d, dynamical system #3, A's roto-translated local coordinate frame

Impact of anisotropic filtering We compare anisotropic and isotropic filtering on inD in fig. 5c. Be it with isotropic or anisotropic filters, the local coordinate frames outperform the competitors. That said, anisotropic filters give a clear advantage over isotropic ones.

Impact of rotation in spherical symmetries Even though particles do not have intrinsic orientations, they do have the direction of their velocity. We postulate that roto-translated local coordinate frames allow for more efficient learning, as long as the coordinate frames are consistently invariant. To motivate this, consider the simple case of a 2D system shown in fig. 6a with three objects A, B, C with no intrinsic orientation. In fig. 6a, B is far above left of A, while C is near below right of A. Rotating the system by π, see fig. 6b, A still lies at the same origin, however, C is now in the top left and B is in the bottom right quadrant. Without canonicalizing with respect to rotations, the description of the two systems is very different, and subsequent neural networks will have to learn to account for this underlying symmetry. On the other hand, different dynamical systems should yield different representations. For example, consider the 2 dynamical systems shown in figs. 6b and 6c. While they only differ in A's velocity, their dynamics vary greatly: in fig. 6c, A and B are moving perpendicularly to one another and may crush due to attractive forces. In the canonicalized frame in fig. 6d, the neighbour representations are indeed very different from fig. 6b.

In the end, we care that our inputs are represented consistently (invariantly) if their relative differences (translations or rotations) are the same according to the system at hand, regardless of how we obtain the reference axis for the rotation (intrinsic angular position or another invariant quantity like the angles of acceleration vectors). Thus, applying both translation and rotation transformation helps even for objects with no intrinsic viewpoint and orientation, like point masses. We confirm the hypothesis in an ablation experiment with charged particles, where using only the translation transformation to form local coordinate frames leads to decreased accuracy, see fig. 5d.

6 Conclusion

In this work we introduced LoCS, a method that introduces canonicalized roto-translated local coordinate frames for all objects in interacting dynamical systems formalized in geometric graphs. These coordinate frames grant us global invariance to roto-translations and naturally allow for anisotropic continuous filtering. We demonstrate the effectiveness of our method in a range of 2D and 3D settings, outperforming recent state-of-the-art works.

Limitations Many dynamical systems in nature are not formalized as geometric graphs (*e.g.* social networks), or are not intrinsically described by angular positions (*e.g.* elementary particles), in which case the proposed method is not applicable. Furthermore, although we approximate angular positions using velocities, our method guarantees full invariance/equivariance to global roto-translations only in the 2D case, while in 3 dimensions we have an equivariance leakage. We have identified two modes where local coordinate frames do not exhibit the same large improvements. First, when the data setting relies, in fact, on a global coordinate frame, like different positions in a basketball court [56]. Second, when the direction of the local coordinate frames cannot be well-defined, like objects with $\mathbf{0}$ velocity (although simply and arbitrarily setting the orientation to $\mathbf{0}$ works just as well). Last, a theoretical limitation is that Euler angles, which LoCS also uses, are prone to singularities and gimbal lock, although in practice we observed no problem.

Acknowledgments

The project is funded by the NWO LIFT grant 'FLORA'.

References

[1] Alahi, A., Goel, K., Ramanathan, V., Robicquet, A., Li, F., and Savarese, S. Social LSTM: Human Trajectory Prediction in Crowded Spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Bahdanau, D., Cho, K., and Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[3] Battaglia, P. W., Pascanu, R., Lai, M., Rezende, D. J., and Kavukcuoglu, K. Interaction Networks for Learning about Objects, Relations and Physics. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.

[4] Bock, J., Krajewski, R., Moers, T., Runde, S., Vater, L., and Eckstein, L. The inD dataset: A drone dataset of naturalistic road user trajectories at german intersections. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020.

[5] Calafiura, P., Farrell, S., Gray, H., Vlimant, J.-R., Innocente, V., Salzburger, A., Amrouche, S., Golling, T., Kiehn, M., Estrade, V., et al. TrackML: A High Energy Physics Particle Tracking Challenge. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, 2018.

[6] Chang, M., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., and Hays, J. Argoverse: 3D Tracking and Forecasting With Rich Maps. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[7] Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[8] Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. A Recurrent Latent Variable Model for Sequential Data. In *Advances in Neural Information Processing Systems 28 (NIPS)*, 2015.

[9] Clevert, D., Unterthiner, T., and Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In *4th International Conference on Learning Representations (ICLR)*, 2016.

[10] CMU. Carnegie-Mellon Motion Capture Database, 2003. URL `http://mocap.cs.cmu.edu`.

[11] Cohen, T. and Welling, M. Group Equivariant Convolutional Networks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

[12] Cohen, T. S. and Welling, M. Steerable CNNs. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[13] de Haan, P., Weiler, M., Cohen, T., and Max, W. Gauge equivariant mesh CNNs: Anisotropic convolutions on geometric graphs. In *9th International Conference on Learning Representations (ICLR)*, 2021.

[14] Deng, H., Birdal, T., and Ilic, S. Ppf-foldnet: Unsupervised learning of rotation invariant 3d local descriptors. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[15] Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

[16] Fey, M., Lenssen, J. E., Weichert, F., and Müller, H. SplineCNN: Fast Geometric Deep Learning With Continuous B-Spline Kernels. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[17] Fuchs, F., Worrall, D. E., Fischer, V., and Welling, M. SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks. In *Advances in Neural Information Processing Systems 33 (NeurIPS)*, 2020.

[18] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural Message Passing for Quantum Chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[19] Graber, C. and Schwing, A. G. Dynamic Neural Relational Inference. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[20] Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive Representation Learning on Large Graphs. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

[21] Helbing, D. and Molnar, P. Social force model for pedestrian dynamics. *Physical review E*, 51(5), 1995.

[22] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8), 1997.

[23] Jang, E., Gu, S., and Poole, B. Categorical Reparameterization with Gumbel-Softmax. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[24] Jia, X., Brabandere, B. D., Tuytelaars, T., and Gool, L. V. Dynamic Filter Networks. In *Advances in Neural Information Processing Systems 29 (NIPS)*, 2016.

[25] Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations (ICLR)*, 2014.

[26] Kipf, T. N. and Welling, M. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[27] Kipf, T. N., Fetaya, E., Wang, K., Welling, M., and Zemel, R. S. Neural Relational Inference for Interacting Systems. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[28] Kosaraju, V., Sadeghian, A., Martín-Martín, R., Reid, I. D., Rezatofighi, H., and Savarese, S. Social-BiGAT: Multimodal Trajectory Forecasting using Bicycle-GAN and Graph Attention Networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[29] Kothari, P., Kreiss, S., and Alahi, A. Human trajectory forecasting in crowds: A deep learning perspective. *arXiv preprint arXiv:2007.03639*, 2020.

[30] Krishnan, R. G., Shalit, U., and Sontag, D. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[31] Lerner, A., Chrysanthou, Y., and Lischinski, D. Crowds by example. In *Computer graphics forum*, volume 26. Wiley Online Library, 2007.

[32] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. Gated Graph Sequence Neural Networks. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[33] Ma, Y., Zhu, X., Zhang, S., Yang, R., Wang, W., and Manocha, D. TrafficPredict: Trajectory Prediction for Heterogeneous Traffic-Agents. In *The Thirty-Third Conference on Artificial Intelligence (AAAI)*, 2019.

[34] Maddison, C. J., Mnih, A., and Teh, Y. W. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *5th International Conference on Learning Representations (ICLR)*, 2017.

[35] Mohamed, A., Qian, K., Elhoseiny, M., and Claudel, C. G. Social-STGCNN: A Social Spatio-Temporal Graph Convolutional Neural Network for Human Trajectory Prediction. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[36] Monti, F., Boscaini, D., Masci, J., Rodolà, E., Svoboda, J., and Bronstein, M. M. Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[37] Pellegrini, S., Ess, A., Schindler, K., and Gool, L. V. You'll never walk alone: Modeling social behavior for multi-target tracking. In *IEEE 12th International Conference on Computer Vision (ICCV)*, 2009.

[38] Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, 2014.

[39] Salzmann, T., Ivanovic, B., Chakravarty, P., and Pavone, M. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *European Conference on Computer Vision (ECCV)*, 2020.

[40] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. W. Learning to Simulate Complex Physics with Graph Networks. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, 2020.

[41] Satorras, V. G., Hoogeboom, E., and Welling, M. E(n) Equivariant Graph Neural Networks. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[42] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE transactions on neural networks*, 20(1), 2008.

[43] Schöller, C., Aravantinos, V., Lay, F., and Knoll, A. What the constant velocity model can teach us about pedestrian motion prediction. *IEEE Robotics and Automation Letters*, 5(2), 2020.

[44] Schütt, K., Kindermans, P., Felix, H. E. S., Chmiela, S., Tkatchenko, A., and Müller, K. SchNet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017.

[45] Simonovsky, M. and Komodakis, N. Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[46] Thomas, N., Smidt, T., Kearnes, S., Yang, L., Li, L., Kohlhoff, K., and Riley, P. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

[47] Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph Attention Networks. In *6th International Conference on Learning Representations (ICLR)*, 2018.

[48] Vemula, A., Muelling, K., and Oh, J. Social attention: Modeling attention in human crowds. In *2018 IEEE international Conference on Robotics and Automation (ICRA)*, 2018.

[49] Vignac, C., Loukas, A., and Frossard, P. Building powerful and equivariant graph neural networks with message-passing. *arXiv preprint arXiv:2006.15107*, 2020.

[50] Walters, R., Li, J., and Yu, R. Trajectory Prediction using Equivariant Continuous Convolution. In *9th International Conference on Learning Representations (ICLR)*, 2021.

[51] Wang, S., Suo, S., Ma, W., Pokrovsky, A., and Urtasun, R. Deep Parametric Continuous Convolutional Neural Networks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[52] Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5), 2019.

[53] Weiler, M. and Cesa, G. General E(2)-Equivariant Steerable CNNs. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[54] Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. Harmonic Networks: Deep Translation and Rotation Equivariance. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[55] Yan, S., Xiong, Y., and Lin, D. Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition. In *Proceedings of the Thirty-Second Conference on Artificial Intelligence (AAAI)*, 2018.

[56] Yue, Y., Lucey, P., Carr, P., Bialkowski, A., and Matthews, I. Learning fine-grained spatial models for dynamic sports play prediction. In *2014 IEEE international conference on data mining*, 2014.

A Roto-translation invariance

A.1 Rotations in 2 dimensions

In 2-dimensional settings, there exists a single scalar angular position, the yaw angle θ. Following eqs. (3) and (5), we compute the rotation matrices $\mathbf{Q}$, $\mathbf{R}$ and $\tilde{\mathbf{R}}$ as follows:

$$\mathbf{Q}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{11}$$

$$\mathbf{R}(\theta) = \mathbf{Q}(\theta) \oplus \mathbf{Q}(\theta) = \begin{pmatrix} \mathbf{Q}(\theta) & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \mathbf{Q}(\theta) \end{pmatrix} \tag{12}$$

$$\tilde{\mathbf{R}}(\theta) = \mathbf{Q}(\theta) \oplus \mathbf{Q}(\theta) \oplus \mathbf{Q}(\theta) = \begin{pmatrix} \mathbf{Q}(\theta) & & \mathbf{0} \\ & \mathbf{Q}(\theta) & \\ \mathbf{0} & & \mathbf{Q}(\theta) \end{pmatrix} \tag{13}$$

The second rotation matrix $\mathbf{Q}$ in $\tilde{\mathbf{R}}$ is used to rotate the angular positions. In order to perform the transformation, we have to express the angular positions in a format suitable for linear transformations; we do so by transforming them to rotation matrices, perform a matrix multiplication, and then transform the angular positions back to angle format. In 2 dimensions, we use eq. (11) to convert the angular positions θ to matrix format. After the rotation, we can convert them back to angle format using the 2-argument arc-tangent function:

$$\theta = \mathrm{atan2}(\sin\theta, \cos\theta) \tag{14}$$

Simplified rotations In 2 dimensions, the computations can be simplified since rotations commute. First, we show that chained rotations result in angle addition/subtraction, that is:

$$\mathbf{Q}(\theta_i) \cdot \mathbf{Q}(\theta_j) = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_j & -\sin\theta_j \\ \sin\theta_j & \cos\theta_j \end{pmatrix} \tag{15}$$

$$= \begin{pmatrix} \cos\theta_i\cos\theta_j - \sin\theta_i\sin\theta_j & -\cos\theta_i\sin\theta_j - \sin\theta_i\cos\theta_j \\ \sin\theta_i\cos\theta_j + \cos\theta_i\sin\theta_j & -\sin\theta_i\sin\theta_j + \cos\theta_i\cos\theta_j \end{pmatrix} \tag{16}$$

$$= \begin{pmatrix} \cos(\theta_i+\theta_j) & -\sin(\theta_i+\theta_j) \\ \sin(\theta_i+\theta_j) & \cos(\theta_i+\theta_j) \end{pmatrix} \tag{17}$$

$$= \mathbf{Q}(\theta_i + \theta_j) \tag{18}$$

Following the same approach, we compute the inverse rotation:

$$\mathbf{Q}^\top(\theta_i) \cdot \mathbf{Q}(\theta_j) = \mathbf{Q}(-\theta_i) \cdot \mathbf{Q}(\theta_j) = \mathbf{Q}(\theta_j - \theta_i) \tag{19}$$

Thus, instead of rotating the angular positions (expressed in rotation matrix form) using the rotation matrix $\mathbf{Q}$, in practice we perform the transformation directly to the angles via addition/subtraction, and replace the matrix $\mathbf{Q}$ with the identity matrix $\mathbf{I}_{1\times 1}$. This results in the following equations that replace eqs. (3) and (4):

$$\tilde{\mathbf{R}}(\theta) = \mathbf{Q}(\theta) \oplus \mathbf{I}_{1\times 1} \oplus \mathbf{Q}(\theta) = \begin{pmatrix} \mathbf{Q}(\theta) & & \mathbf{0} \\ & \mathbf{I}_{1\times 1} & \\ \mathbf{0} & & \mathbf{Q}(\theta) \end{pmatrix} \tag{20}$$

$$\mathbf{v}^t_{j|i} = \tilde{\mathbf{R}}^{t\top}_i \left[\mathbf{r}^t_{j,i}, \theta^t_j - \theta^t_i, \mathbf{u}^t_j\right] \tag{21}$$

Angular position approximation In order to approximate the yaw angle θ using the velocity vector $\mathbf{u} = (u_x, u_y)^\top$, we transform the velocities to polar coordinates and use the azimuth angle of the polar representation to compute θ as follows:

$$\theta = \tan^{-1}\left(\frac{u_y}{u_x}\right) \tag{22}$$

In practice, we use the 2-argument arc-tangent function $\mathrm{atan2}(y, x)$ to compute θ.

Computing the relative angular position can result in angles outside the range $[-\pi, \pi)$, which can lead to discrepancies. Thus, we wrap the computed angle difference so that it always belongs in that range. Furthermore, in all cases that angles are not used geometrically (*e.g.* for rotations), we normalize them by dividing by π, resulting in an output range of $[-1, 1)$.

A.2 Rotations 3 dimensions

In 3 dimensions, the computation of rotation matrices is more involved than the 2D case. As described in section 3.1, we decompose the rotation matrix $\mathbf{Q}(\boldsymbol{\omega})$ into 3 chained elemental rotations $\mathbf{Q}_z(\theta)$, $\mathbf{Q}_y(\phi)$ and $\mathbf{Q}_x(\psi)$. The elemental rotation matrices are computed as follows:

$$\mathbf{Q}_z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{23}$$

$$\mathbf{Q}_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \tag{24}$$

$$\mathbf{Q}_x(\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{pmatrix} \tag{25}$$

Next, we compose the elemental matrices to compute the full rotation matrix:

$$\mathbf{Q}(\boldsymbol{\omega}) = \mathbf{Q}_z(\theta)\mathbf{Q}_y(\phi)\mathbf{Q}_x(\psi) \tag{26}$$

$$= \begin{pmatrix} \cos\phi\cos\theta & \sin\psi\sin\phi\cos\theta - \cos\psi\sin\theta & \cos\psi\sin\phi\cos\theta + \sin\psi\sin\theta \\ \cos\phi\sin\theta & \sin\psi\sin\phi\sin\theta + \cos\psi\cos\theta & \cos\psi\sin\phi\sin\theta - \sin\psi\cos\theta \\ -\sin\phi & \sin\psi\cos\phi & \cos\psi\cos\phi \end{pmatrix} \tag{27}$$

$$\mathbf{Q}^\top(\boldsymbol{\omega}) = \mathbf{Q}_x^\top(\psi)\mathbf{Q}_y^\top(\phi)\mathbf{Q}_z^\top(\theta) \tag{28}$$

$$= \begin{pmatrix} \cos\phi\cos\theta & \cos\phi\sin\theta & -\sin\phi \\ \sin\psi\sin\phi\cos\theta - \cos\psi\sin\theta & \sin\psi\sin\phi\sin\theta + \cos\psi\cos\theta & \sin\psi\cos\phi \\ \cos\psi\sin\phi\cos\theta + \sin\psi\sin\theta & \cos\psi\sin\phi\sin\theta - \sin\psi\cos\theta & \cos\psi\cos\phi \end{pmatrix} \tag{29}$$

$$\mathbf{R}(\boldsymbol{\omega}) = \begin{pmatrix} \mathbf{Q}(\boldsymbol{\omega}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{Q}(\boldsymbol{\omega}) \end{pmatrix} \tag{30}$$

$$\tilde{\mathbf{R}}(\boldsymbol{\omega}) = \begin{pmatrix} \mathbf{Q}(\boldsymbol{\omega}) & & \mathbf{0} \\ & \mathbf{Q}(\boldsymbol{\omega}) & \\ \mathbf{0} & & \mathbf{Q}(\boldsymbol{\omega}) \end{pmatrix} \tag{31}$$

Similar to the 2D case, in order to rotate the angular positions we have to convert them to a format suitable for linear transformations. We use eqs. (26) and (27) to perform the conversion. After rotation, we convert the angular positions back to angle format. Using $\boldsymbol{\Omega}$ to denote the transformed angular positions expressed in matrix format, we have the following:

$$\boldsymbol{\omega} = \begin{pmatrix} \theta \\ \phi \\ \psi \end{pmatrix} = \begin{pmatrix} \mathrm{atan2}(\boldsymbol{\Omega}_{1,0}, \boldsymbol{\Omega}_{0,0}) \\ \sin^{-1}(-\boldsymbol{\Omega}_{2,0}) \\ \mathrm{atan2}(\boldsymbol{\Omega}_{2,1}, \boldsymbol{\Omega}_{2,2}) \end{pmatrix} \tag{32}$$

Angular position approximation Using the velocity angles to approximate angular positions and create the local coordinate frames in 3 dimensions is not as straight-forward as the 2-dimensional case. The spherical coordinates representation of the velocity vector gives us 2 angles instead of the 3 that are required to fully describe a 6-DOF 3D rigid body.

In the following equations, we use the notation convention (ρ, θ, ϕ) to represent the radial distance, azimuthal angle and polar angle, respectively. The transformations from Cartesian to spherical coordinates are as follows:

$$\rho = \sqrt{u_x^2 + u_y^2 + u_z^2} \tag{33}$$

$$\theta = \tan^{-1}\left(\frac{u_y}{u_x}\right) \tag{34}$$

$$\phi = \cos^{-1}\left(\frac{u_z}{\rho}\right) \tag{35}$$

In practice, similar to the 2-dimensional setting, we use the $\mathrm{atan2}$ function to compute θ. Furthermore, we add $\epsilon = 1e-8$ to the denominator in eq. (35) and clamp the fraction in the range $[-1, 1]$ to avoid numerical instabilities that may occur, especially during backpropagation.

Having access to 2 angular positions, we compute the rotation matrix $\mathbf{Q}$ as follows:

$$\mathbf{Q}(\boldsymbol{\omega}) \overset{\psi=0}{=} \mathbf{Q}_z(\theta)\mathbf{Q}_y(\phi) = \begin{pmatrix} \cos\phi\cos\theta & -\sin\theta & \sin\phi\cos\theta \\ \cos\phi\sin\theta & \cos\theta & \sin\phi\sin\theta \\ -\sin\phi & 0 & \cos\phi \end{pmatrix} \tag{36}$$

$$\mathbf{Q}^\top(\boldsymbol{\omega}) \overset{\psi=0}{=} \mathbf{Q}_y^\top(\phi)\mathbf{Q}_z^\top(\theta) = \begin{pmatrix} \cos\phi\cos\theta & \cos\phi\sin\theta & -\sin\phi \\ -\sin\theta & \cos\theta & 0 \\ \sin\phi\cos\theta & \sin\phi\sin\theta & \cos\phi \end{pmatrix} \tag{37}$$

Finally, similar to the 2-dimensional setting, we normalize relative angular positions so that their output range is $[-1, 1)$.

A.3 Proof of roto-translation invariance

Our method explicitly infers the graph structure over a discrete latent graph and simultaneously learns the dynamical system. Learning the graph structure is a roto-translation invariant task; we want to predict the same edge distribution for each pair of vertices regardless of the global rotation of translation. On the other hand, trajectory forecasting is a roto-translation equivariant task; a global translation and rotation to the input trajectories should affect the output trajectories equivalently. In this section, we derive the proof on roto-translation invariance/equivariance.

Let $\mathbf{Q}_g \in \mathbb{R}^{D\times D}$ be a global rotation matrix in D dimensions and $\boldsymbol{\tau}_g \in \mathbb{R}^{D\times 1}$ be a global translation vector. As explained in section 2.1, input trajectories are described by the states $\mathbf{x}_i^t = [\mathbf{p}_i^t, \mathbf{u}_i^t]$. Similarly, we use $\mathbf{v}_i^t = [\mathbf{p}_i^t, \boldsymbol{\omega}_i^t, \mathbf{u}_i^t]$ to denote the augmented states, described by the linear position, angular position and linear velocity. Finally, we introduce the notation $\mathbf{X}$ and $\mathbf{V}$ to denote the set of states and augmented states, respectively, organized in matrix form.

In the following equations, we remove time indices to reduce clutter. Similar to eq. (3) we define the matrices $\mathbf{R}_g$ and $\tilde{\mathbf{R}}_g$. We have:

$$\mathbf{R}_g = \mathbf{Q}_g \oplus \mathbf{Q}_g \tag{38}$$

$$\tilde{\mathbf{R}}_g = \mathbf{Q}_g \oplus \mathbf{Q}_g \oplus \mathbf{Q}_g \tag{39}$$

Equivalently, we define the augmented translation vectors $\boldsymbol{\delta}_g$ and $\tilde{\boldsymbol{\delta}}_g$:

$$\boldsymbol{\delta}_g = [\boldsymbol{\tau}_g, \mathbf{0}_D] \tag{40}$$

$$\tilde{\boldsymbol{\delta}}_g = [\boldsymbol{\tau}_g, \mathbf{0}_D, \mathbf{0}_D] \tag{41}$$

The definition above holds because velocities and angular positions are translation invariant.

First, we will prove that the transformation to the local coordinate systems is invariant to global translations and rotations. Let J denote the function that converts the augmented states to the local coordinate frames. It is formulated as follows:

$$\mathbf{v}_{j|i} = \mathrm{J}(\mathbf{V})_j \tag{42}$$

$$= \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i)[\mathbf{p}_j - \mathbf{p}_i, \boldsymbol{\omega}_j, \mathbf{u}_j] \tag{43}$$

$$= \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i)[\mathbf{r}_{j,i}, \boldsymbol{\omega}_j, \mathbf{u}_j] \tag{44}$$

Local coordinate frames translation invariance To prevent the notation from clutter, in the following equations, we will slightly abuse mathematical notation and use the convention $\mathbf{V} + \boldsymbol{\delta}_g$ to denote the translation of each augmented state in $\mathbf{V}$. Programmatically, we can say that we broadcast $\boldsymbol{\delta}_g$ to match the size of $\mathbf{V}$.

$$\mathrm{J}(\mathbf{V} + \boldsymbol{\delta}_g)_j = \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i)[\mathbf{p}_j + \boldsymbol{\tau}_g - (\mathbf{p}_i - \boldsymbol{\tau}_g), \boldsymbol{\omega}_j, \mathbf{u}_j] \tag{45}$$

$$= \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i)[\mathbf{r}_{j,i}, \boldsymbol{\omega}_j, \mathbf{u}_j] \tag{46}$$

$$= \mathrm{J}(\mathbf{V})_j \tag{47}$$

Local coordinate frames rotation invariance For the canonicalization of the local coordinate systems, we use the matrices $\tilde{\mathbf{R}}_i = \tilde{\mathbf{R}}(\boldsymbol{\omega}_i)$. These matrices transform under global rotation via the following transformation:

$$\hat{\mathbf{R}}_i = \mathbf{R}_g \cdot \mathbf{R}(\boldsymbol{\omega}_i) \tag{48}$$

$$\hat{\mathbf{R}}_i^\top = \mathbf{R}^\top(\boldsymbol{\omega}_i) \cdot \mathbf{R}_g^\top \tag{49}$$

$$\hat{\tilde{\mathbf{R}}}_i = \tilde{\mathbf{R}}_g \cdot \tilde{\mathbf{R}}(\boldsymbol{\omega}_i) \tag{50}$$

$$\hat{\tilde{\mathbf{R}}}_i^\top = \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i) \cdot \tilde{\mathbf{R}}_g^\top \tag{51}$$

Then, we proceed as follows:

$$\mathrm{J}\Big(\tilde{\mathbf{R}}_g \cdot \mathbf{V}\Big)_j = \hat{\tilde{\mathbf{R}}}^\top(\boldsymbol{\omega}_i) \cdot [\mathbf{Q}_g \cdot \mathbf{p}_j - \mathbf{Q}_g \cdot \mathbf{p}_i, \mathbf{Q}_g \cdot \boldsymbol{\omega}_j, \mathbf{Q}_g \cdot \mathbf{u}_j] \tag{52}$$

$$= \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i) \cdot \tilde{\mathbf{R}}_g^\top \cdot [\mathbf{Q}_g \cdot \mathbf{r}_{j,i}, \mathbf{Q}_g \cdot \boldsymbol{\omega}_j, \mathbf{Q}_g \cdot \mathbf{u}_j] \tag{53}$$

$$= \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i) \cdot \tilde{\mathbf{R}}_g^\top \cdot \tilde{\mathbf{R}}_g \cdot [\mathbf{r}_{j,i}, \boldsymbol{\omega}_j, \mathbf{u}_j] \tag{54}$$

$$= \tilde{\mathbf{R}}^\top(\boldsymbol{\omega}_i) \cdot [\mathbf{r}_{j,i}, \boldsymbol{\omega}_j, \mathbf{u}_j] \tag{55}$$

$$= \mathrm{J}(\mathbf{V})_j \tag{56}$$

Encoder roto-translation invariance Next, we will prove that the encoder is rotation and translation invariant. Let E denote the encoder. The encoder takes as inputs the set of roto-translated augmented states $\mathbf{V}_{\text{local}} = \big\{\mathbf{v}_{j|i} \mid j, i \in \{1, \dots, N\}\big\}$. We have already proven that these inputs are invariant to global translations and rotations. Thus, it follows that the encoder is also roto-translation invariant.

Decoder roto-translation equivariance The decoder takes as inputs the set of roto-translated augmented states $\mathbf{V}_{\text{local}} = \big\{\mathbf{v}_{j|i} \mid j, i \in \{1, \dots, N\}\big\}$ as well as the predicted latent edges $\mathbf{z}_{j,i}$. We use $\mathbf{Z} = \{\mathbf{z}_{j,i} \mid j, i \in \{1, \dots, N\}, j \neq i\}$ to denote the set of all latent edges.

To prove that the decoder is equivariant to global rotations and translations, we will split its functionality into 2 consecutive components. Let G be the first component that predicts the differences in position and velocity $\boldsymbol{\Delta}\mathbf{x}$ in the local coordinate systems. Let H be the second component that transforms the predictions from the local coordinate systems to the global coordinate system, as described by eq. (70). The first part of the decoder takes as inputs the augmented states $\mathbf{V}_{\text{local}}$ as well as the latent edges $\mathbf{Z}$. $\mathbf{Z}$ is the output of the encoder, and as we proved earlier, it is invariant. $\mathbf{V}_{\text{local}}$ is also invariant. Hence, G is roto-translation invariant.

Finally, we have to prove that H is equivariant to global translations and rotations. H is a function of $\mathbf{X}$ and $\mathbf{V}_{\text{local}}$ and is defined as $\mathrm{H}(\mathbf{X}, \mathbf{V}_{\text{local}})_i = \mathbf{x}_i + \mathbf{R}(\boldsymbol{\omega}_i) \cdot \mathrm{G}(\mathbf{V}_{\text{local}})_i$.

First, we will prove that H is translation equivariant. We have the following:

$$\mathrm{H}(\mathbf{X}, \mathbf{V}_{\text{local}})_i = \mathbf{x}_i + \mathbf{R}(\boldsymbol{\omega}_i) \cdot \mathrm{G}(\mathbf{V}_{\text{local}})_i \tag{57}$$

$$\mathrm{H}\Big(\mathbf{X} + \boldsymbol{\delta}_g, \mathbf{V}_{\text{local}} + \tilde{\boldsymbol{\delta}}_g\Big)_i = \mathbf{x}_i + \boldsymbol{\delta}_g + \mathbf{R}(\boldsymbol{\omega}_i) \cdot \mathrm{G}\Big(\mathbf{V}_{\text{local}} + \tilde{\boldsymbol{\delta}}_g\Big)_i \tag{58}$$

$$= \mathbf{x}_i + \boldsymbol{\delta}_g + \mathbf{R}(\boldsymbol{\omega}_i) \cdot \mathrm{G}(\mathbf{V}_{\text{local}})_i \tag{59}$$

$$= \mathrm{H}(\mathbf{X}, \mathbf{V}_{\text{local}})_i + \boldsymbol{\delta}_g \tag{60}$$

Next, we will prove that H is rotation equivariant. We have the following:

$$\mathrm{H}\Big(\mathbf{R}_g \cdot \mathbf{X}, \tilde{\mathbf{R}}_g \cdot \mathbf{V}_{\text{local}}\Big)_i = \mathbf{R}_g \cdot \mathbf{x}_i + \mathbf{R}_g \cdot \mathbf{R}(\boldsymbol{\omega}_i) \cdot \mathrm{G}\Big(\tilde{\mathbf{R}}_g \cdot \mathbf{V}_{\text{local}}\Big)_i \tag{61}$$

$$= \mathbf{R}_g \cdot (\mathbf{x}_i + \mathbf{R}(\boldsymbol{\omega}_i) \cdot \mathrm{G}(\mathbf{V}_{\text{local}})_i) \tag{62}$$

$$= \mathbf{R}_g \cdot \mathrm{H}(\mathbf{X}, \mathbf{V}_{\text{local}})_i \tag{63}$$

B Implementation details

B.1 LoCS

Encoder & Prior The embeddings $\mathbf{h}_{j,i}^{(2)}$ are fed into 2 LSTMs [22], one forward in time that computes the prior and one backwards in time for the encoder. The hidden state from the forward LSTM is used to compute the prior distribution, while the hidden states from both the forward and the backward LSTM are concatenated to compute the encoder distribution, according to the following equations:

$$\mathbf{h}_{(j,i),\text{prior}}^{t} = \text{LSTM}_{\text{prior}}\Big(\mathbf{h}_{j,i}^{(2)}, \mathbf{h}_{(j,i),\text{prior}}^{t-1}\Big) \tag{64}$$

$$\mathbf{h}_{(j,i),\text{enc}}^{t} = \text{LSTM}_{\text{enc}}\Big(\mathbf{h}_{j,i}^{(2)}, \mathbf{h}_{(j,i),\text{enc}}^{t+1}\Big) \tag{65}$$

$$p_\phi\big(\mathbf{z}^t|\mathbf{x}^{1:t}, \mathbf{z}^{1:t-1}\big) = \text{softmax}\Big(f_{\text{prior}}\Big(\mathbf{h}_{(j,i),\text{prior}}^{t}\Big)\Big) \tag{66}$$

$$q_\phi\big(\mathbf{z}_{j,i}^t|\mathbf{x}\big) = \text{softmax}\Big(f_{\text{enc}}\Big(\Big[\mathbf{h}_{(j,i),\text{prior}}^{t}, \mathbf{h}_{(j,i),\text{enc}}^{t}\Big]\Big)\Big) \tag{67}$$

The functions $f_{\text{enc}}, f_{\text{prior}}$ are MLPs that map the hidden states to $\mathbb{R}^K$, where K is the number of latent edge types.

Decoder Following [19, 27], we use 2 different decoders based on whether the governing dynamics are Markovian. In both cases, the decoders have similar structure with [19, 27]; the main difference is that we operate entirely on the roto-translated local coordinate frames. In order to convert our predictions back to the global coordinate frame, we perform an inverse rotation by $\mathbf{R}_i^t = \mathbf{R}(\boldsymbol{\omega}_i^t) = \mathbf{Q}(\boldsymbol{\omega}_i^t) \oplus \mathbf{Q}(\boldsymbol{\omega}_i^t)$.

Markovian decoder In many applications, such as dynamical systems in physics, the governing dynamics satisfy the Markov property $p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t}) = p_\theta(\mathbf{x}^{t+1}|\mathbf{x}^t, \mathbf{z}^t)$. In this case, we use the following decoder:

$$\mathbf{m}_{j,i}^t = \sum_k z_{(j,i),k}^t f^k\Big(\Big[\mathbf{v}_{j|i}^t, \mathbf{v}_{i|i}^t\Big]\Big) \tag{68}$$

$$\mathbf{m}_i^t = f_v^{(3)}\left(g_v^{(3)}\Big(\mathbf{v}_{i|i}^t\Big) + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\mathbf{m}_{j,i}^t\right) \tag{69}$$

$$\boldsymbol{\mu}_i^{t+1} = \mathbf{x}_i^t + \mathbf{R}_i^t \cdot f_v^{(4)}\big(\mathbf{m}_i^t\big) \tag{70}$$

$$p(\mathbf{x}_i^{t+1}|\mathbf{x}^t, \mathbf{z}^t) = \mathcal{N}\big(\boldsymbol{\mu}_i^{t+1}, \sigma^2\mathbf{I}\big) \tag{71}$$

The functions $f_v^{(3)}, f_v^{(4)}$ and $f^k, k \in \{1, \dots K\}$ are MLPs, while $g_v^{(3)}$ is a linear layer. The output of the model is the mean estimate of a multivariate isotropic Gaussian distribution with fixed variance.

Recurrent decoder In most real-world applications, the Markovian assumption does not hold. In this case we use a recurrent decoder.

$$\mathbf{m}_{j,i}^t = \sum_k z_{(j,i),k}^t f^k\Big(\Big[\mathbf{v}_{j|i}^t, \mathbf{v}_{i|i}^t\Big]\Big) \tag{72}$$

$$\mathbf{m}_i^t = f_v^{(3)}\left(g_v^{(3)}\Big(\mathbf{v}_{i|i}^t\Big) + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\mathbf{m}_{j,i}^t\right) \tag{73}$$

$$\mathbf{h}_{j,i}^t = \sum_k z_{(j,i),k}^t g^k\big(\big[\mathbf{h}_j^t, \mathbf{h}_i^t\big]\big) \tag{74}$$

$$\mathbf{n}_i^t = \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\mathbf{h}_{(j,i)}^t \tag{75}$$

$$\mathbf{h}_i^{t+1} = \text{GRU}\big(\big[\mathbf{n}_i^t, \mathbf{m}^t\big], \mathbf{h}_i^t\big) \tag{76}$$

$$\boldsymbol{\mu}_i^{t+1} = \mathbf{x}_i^t + \mathbf{R}_i^t \cdot f_v^{(4)}\big(\mathbf{h}_i^{t+1}\big) \tag{77}$$

$$p(\mathbf{x}_i^{t+1}|\mathbf{x}^{1:t}, \mathbf{z}^{1:t}) = \mathcal{N}\big(\boldsymbol{\mu}_i^{t+1}, \sigma^2\mathbf{I}\big) \tag{78}$$

The functions $g^k, k \in \{1, \dots K\}$ are MLPs. The GRU block [7] is identical to the one used in [27].

Loss Following [19], we train our models by minimizing the Evidence Lower Bound (ELBO), which comprises the reconstruction loss of the predicted trajectories (positions and velocities) and the KL divergence.

$$\mathcal{L}(\phi, \theta) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}|\mathbf{x})] \tag{79}$$

As mentioned earlier, we assume the outputs follow an isotropic Gaussian distribution with fixed variance. The reconstruction loss and the KL divergence take the following form:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] = -\sum_i \sum_t \frac{||\mathbf{x}_i^t - \boldsymbol{\mu}_i^t||}{2\sigma^2} + \frac{1}{2}\log\big(2\pi\sigma^2\big) \tag{80}$$

$$\mathrm{KL}[q_\phi(\mathbf{z}|\mathbf{x})||p_\phi(\mathbf{z}|\mathbf{x})] = \sum_{t=1}^{T}\left(\mathbb{H}(q_\phi(\mathbf{z}_{ji}^t|\mathbf{x})) - \sum_{\mathbf{z}_{ji}^t} q_\phi(\mathbf{z}_{ji}^t|\mathbf{x}) \log p_\theta(\mathbf{z}_{ji}^t|\mathbf{x}^{1:t}, \mathbf{z}^{1:t-1})\right) \tag{81}$$

$\mathbb{H}$ denotes the entropy operator. In all experiments, we set $\sigma^2 = 10^{-5}$.

Spherical coordinate relative positions Many works in the literature [41, 44] employ distance-based message-passing steps and filters as a means to better model interactions. We also find that explicitly incorporating Euclidean distances is useful in practice. We augment the canonicalized states $\mathbf{v}_{j|i}$ with the spherical representations of the relative positions $\mathbf{r}_{j,i}$. We denote the spherical relative positions as $\mathbf{s}_{j,i}$. They are computed as follows:

$$\mathbf{s}_{j,i}^t = \text{cart2spherical}\big(\mathbf{Q}^{t\top}(\boldsymbol{\omega}_i) \cdot \mathbf{r}_{j,i}^t\big) \tag{82}$$

The spherical representations are computed within the roto-translated coordinate frames and thus, have no effect on the roto-translation invariance. We modify eq. (6), omitting the time indices for clarity:

$$\mathbf{h}_{j,i}^{(1)} = f_e^{(1)}\big(\big[\mathbf{v}_{j|i}, \mathbf{s}_{j,i}, \mathbf{v}_{i|i}\big]\big) \tag{83}$$

Similarly, we modify eqs. (68) and (72) as follows:

$$\mathbf{m}_{j,i}^t = \sum_k z_{(j,i),k}^t f^k\Big(\Big[\mathbf{v}_{j|i}^t, \mathbf{s}_{j,i}^t, \mathbf{v}_{i|i}^t\Big]\Big) \tag{84}$$

Anisotropic filtering The anisotropic filters presented in section 3.4 are used to compute the latent edge embeddings. Specifically, we replace the filters in eq. (6) in the encoder. The filter generating network is a 2-layer MLP with ELU [9] activation in the hidden layer. In the inD [4] experiment, we also use anisotropic filters in the decoder. These filters replace the filters in eq. (72). In this case, we use a 2-layer MLP with $\tanh$ activation in the hidden layer. In all experiments, we use the spherical relative positions as input to the filter generating network instead of the Cartesian relative positions. Using $\boldsymbol{\Delta}\mathbf{p}_{j,i}^t = \big[\mathbf{s}_{j,i}^t, \mathbf{Q}^{t\top}(\boldsymbol{\omega}_i) \cdot \boldsymbol{\omega}_j^t\big]$ to denote the canonicalized relative linear and angular positions, the encoder and decoder filter generating networks are formulated as:

$$\mathbf{h}_{j,i}^{(1),t} = \mathbf{W}_{\mathcal{F}}\big(\boldsymbol{\Delta}\mathbf{p}_{j,i}^t\big) \cdot \Big[\mathbf{v}_{j|i}^t, \mathbf{s}_{j,i}^t, \mathbf{v}_{i|i}^t\Big] \tag{85}$$

$$\mathbf{m}_{j,i}^t = \sum_k z_{(j,i),k}^t \mathbf{W}_{\mathcal{F}}^k\big(\boldsymbol{\Delta}\mathbf{p}_{j,i}^t\big) \cdot \Big(\Big[\mathbf{v}_{j|i}^t, \mathbf{s}_{j,i}^t, \mathbf{v}_{i|i}^t\Big]\Big) \tag{86}$$

B.2 NRI & dNRI

We use the official dNRI implementation from `https://github.com/cgraber/cvpr_dNRI`. We use the same repository for its NRI implementation as well.

B.3 EGNN

We use the official EGNN implementation from `https://github.com/vgsatorras/egnn`. In all experiments we use the EGNN model with position and velocity inputs/outputs. Each layer is defined as $\mathbf{h}^{l+1}, \mathbf{p}^{l+1}, \mathbf{u}^{l+1} = \text{EGCL}\big(\mathbf{h}^l, \mathbf{p}^l, \mathbf{u}^l\big)$.

The hidden state at the input layer $\mathbf{h}_i^0$ is computed via a linear layer ψ_h that embeds the input (scalar) speed of each node to the hidden dimension of the model, $\mathbf{h}_i^0 = \psi_h\big(\big\|\mathbf{u}_i^0\big\|\big)$. Each EGNN layer is formulated as follows:

$$\mathbf{m}_{j,i} = \phi_e\Big(\mathbf{h}_i^l, \mathbf{h}_j^l, \big\|\mathbf{p}_j^l - \mathbf{p}_i^l\big\|_2^2\Big) \tag{87}$$

$$\mathbf{u}_i^{l+1} = \phi_v\big(\mathbf{h}_i^l\big)\mathbf{u}_i^l + \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)}\big(\mathbf{p}_j^l - \mathbf{p}_i^l\big)\cdot\phi_x(\mathbf{m}_{j,i}) \tag{88}$$

$$\mathbf{p}_i^{l+1} = \mathbf{p}_i^l + \mathbf{u}_i^{l+1} \tag{89}$$

$$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)}\phi_w(\mathbf{m}_{j,i})\cdot\mathbf{m}_{j,i} \tag{90}$$

$$\mathbf{h}_i^{l+1} = \phi_h\big(\mathbf{h}_i^l, \mathbf{m}_i\big) \tag{91}$$

The functions $\phi_e, \phi_v, \phi_x, \phi_h, \phi_w$ are MLPs with learnable parameters that closely follow the original work. More specifically, the functions ϕ_e, ϕ_v, ϕ_x and ϕ_h are 2-layer MLPs, and the function ϕ_w is a linear layer with a sigmoid activation used to weigh the messages before aggregation.

The EGNN model comprises 4 layers and the hidden dimensions in all layers are 64. The training and evaluation schemes are identical to the other models, except that the model is trained by minimizing the negative log-likelihood of a Gaussian distribution of the positions and velocities, following Equation 80.

B.4 Computing resources

We ran all experiments on internal clusters using single GPU jobs. 3 different GPU models were used in total, namely the Nvidia RTX 2080 Ti, Nvidia GTX 1080 Ti, and Nvidia TitanX. The source code was written in PyTorch [61], version 1.4.0 with CUDA 10.0.

B.5 Hyperparameters & training details

For the synthetic experiment, we follow [19] and train models for 200 epochs. We use 2 edges types and hardcode the first edge type to indicate absence of interactions, with a no-edge prior of 0.9. For the charged particles [27], we use 2 edges types with a uniform prior and train the models for 200 epochs. For inD [4], we follow [19] and train models for 400 epochs. We use 4 edge types and hardcode the first to indicate absence of interactions. For motion capture [10] subject #35, we follow [19] and train models for 600 epochs. We use 4 edge types and hardcode the first to indicate absence of interactions. In all experiments, we train LoCS using Adam [25] with a learning rate of $5\text{e}{-4}$.

Encoder & Prior $f_v^{(1)}$ and $f_e^{(2)}$ are 2-layer MLPs with ELU [9] activations after each layer and Batch Normalization [58] at the end, with 256 hidden and output dimensions. $g_v^{(1)}$ is a linear layer with 256 output dimensions. $\text{LSTM}_{\text{prior}}$ and LSTM_{enc} are LSTMs [22] with 64 hidden dimensions. f_{prior} and f_{enc} are 3-layer MLPs with ELU activations after the first 2 layers, 128 hidden dimensions, and K output dimensions, where K is the number of latent edge types. The filter generating network $\mathbf{W}_{\mathcal{F}}$ is a 2-layer MLP, with ELU after the first layer, 256 hidden dimensions. For the experiment on inD [4], we use 64 hidden dimensions for the filter generating network instead.

Decoder $g_v^{(3)}$ is a linear layer with 256 output dimensions. f_k are 2-layer MLPs with ReLU [60] activations after each layer and g_k are 2-layer MLPs with $\tanh$ activations after each layer. $f_v^{(3)}$ is the identity function and $f_v^{(4)}$ is a 3-layer MLP with ReLU activations after the first 2 layers, 256 hidden dimensions and $2D$ output dimensions.

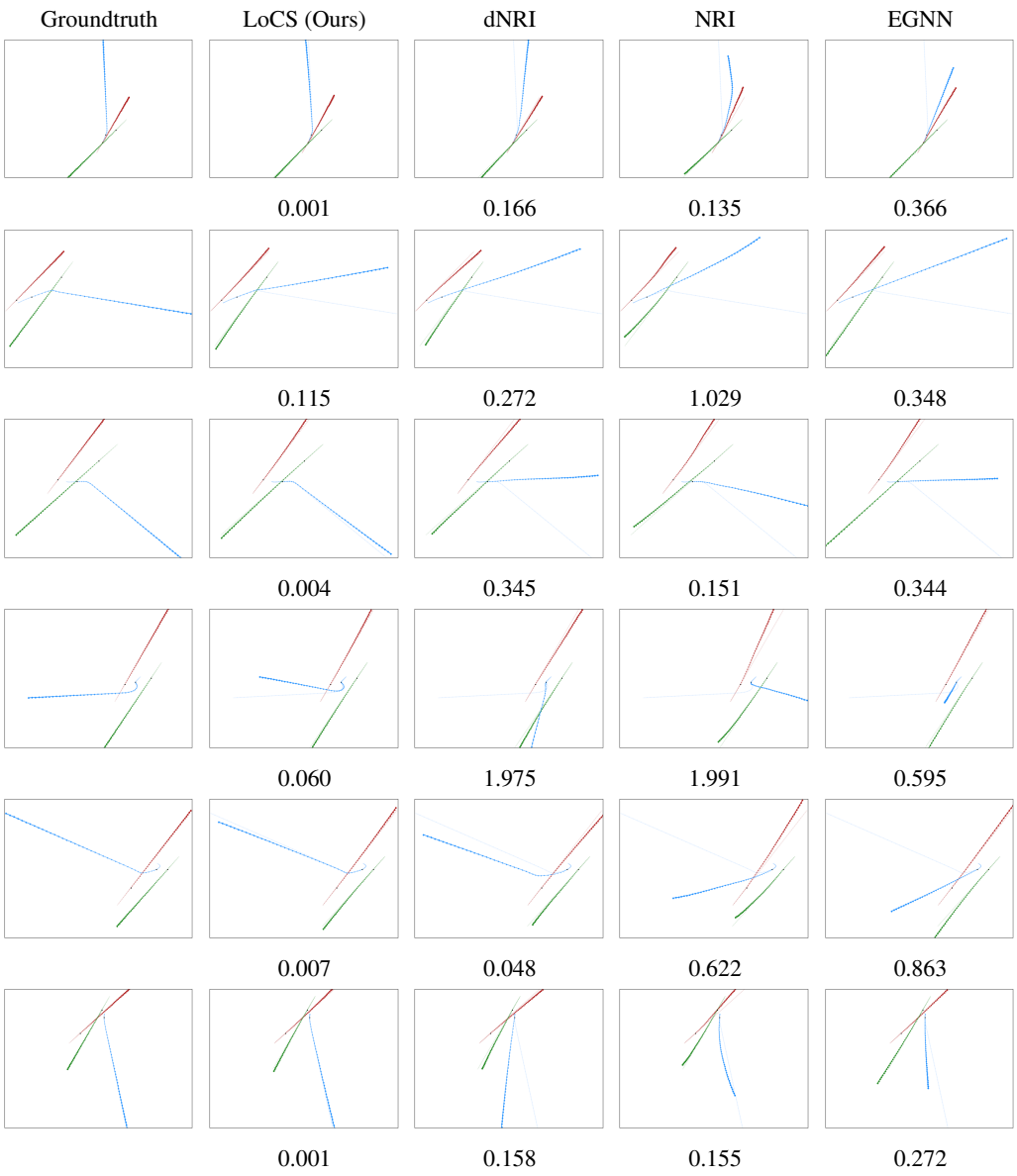


Figure 7: Qualitative results on synthetic dataset, scenes #0 – #5

The filter generating network $\mathbf{W}_{\mathcal{F}}$ is a 2-layer MLP, with $\tanh$ activation after the first layer and 256 hidden dimensions. For the experiment on inD [4], we use 64 hidden dimensions for the filter generating network instead.

The GRU block [7] in the recurrent decoder is identical to the one used in [27], with 256 hidden dimensions.

C Qualitative results

C.1 Synthetic

Figure 7 shows comparative qualitative results for the synthetic dataset [19]. The numbers below each sub-figure represent respective MSE errors.

C.2 Charged

Figure 8 shows comparative qualitative results for 3D charged particles [27]. The numbers below each sub-figure represent respective errors.

C.3 InD

Figure 9 shows comparative qualitative results for inD [4].

C.4 Charged - Interactive

Figure 10 shows comparative qualitative results for the highly interactive subset of 3D charged particles. The numbers below each sub-figure represent respective errors.

D Quantitative results

D.1 Charged particles

Figure 11 shows MSE and L_2 errors for charged particles [27].

D.2 Traffic trajectory forecasting

Figure 12 shows MSE and L_2 errors for inD [4].

D.3 Motion capture

Figure 13 shows MSE and L_2 errors for motion capture [10], subject #35.

D.4 Ablation experiments

The following figures show the complete error curves for the ablation experiments. Figure 14 shows the errors for the highly interactive charged particles subset. Figure 15 shows the results of training dNRI using speed normalization. Figure 16 shows the impact of anisotropic continuous filtering in our method. The roto-translated local coordinate frames already outperform compared methods, while incorporating the anisotropic filters boosts performance even further. Finally, fig. 17 shows the impact of rotation in local coordinate frames, specifically in scenarios without intrinsic orientations, such as charged particles.

E Version history

This is version `v2` of the paper. Compared to version `v1` we have:

- Removed the dependency on PyTorch3D [62] from the source code and updated the manuscript accordingly. More specifically, we have implemented our own `matrix_to_euler_angles` for the `ZYX` convention. Our implementation is a simplification of the aforementioned function and is functionally identical to it.

References

[57] Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[58] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2015.

[59] Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[60] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012.

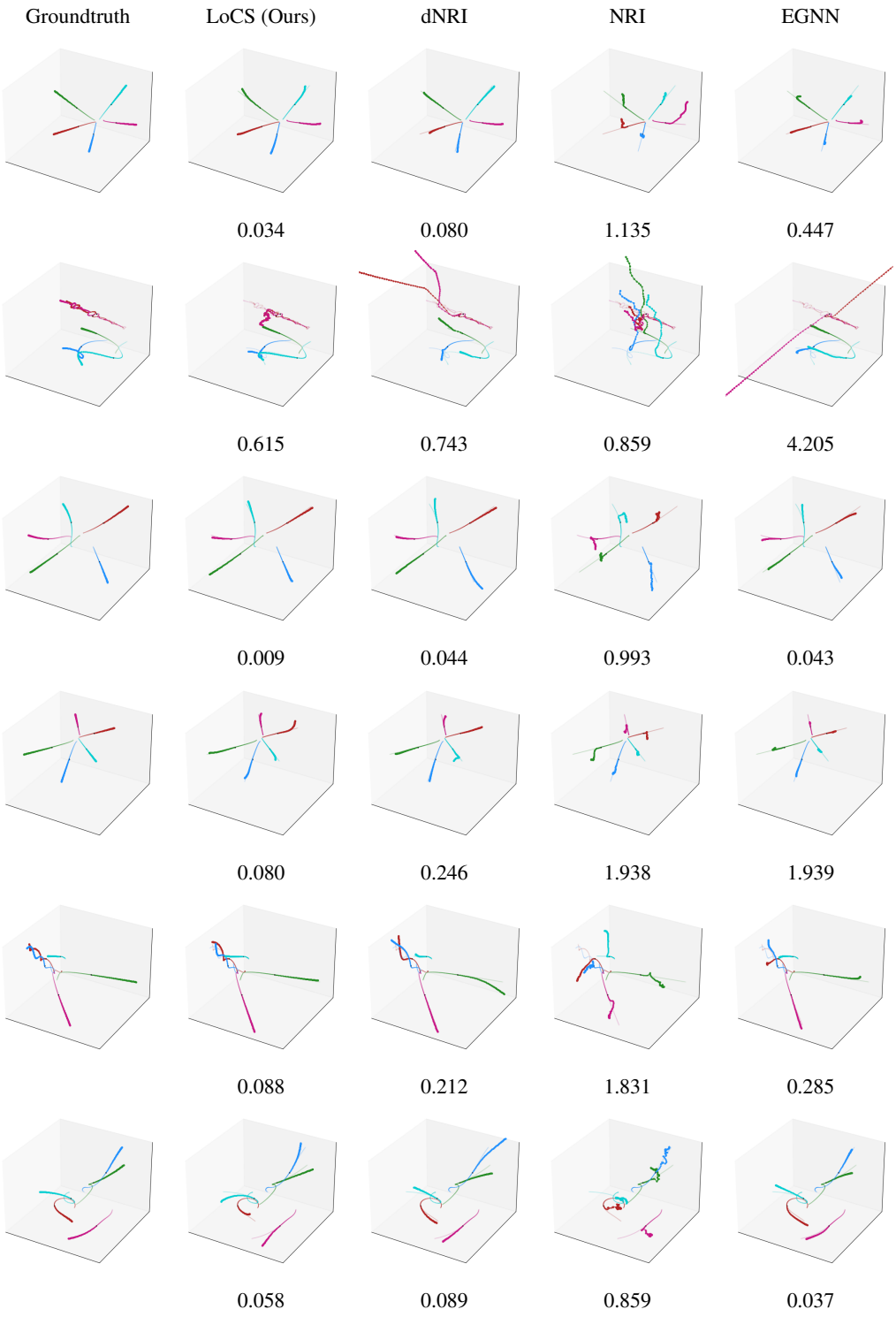


Figure 8: Qualitative results on charged particles, scenes #0 – #5

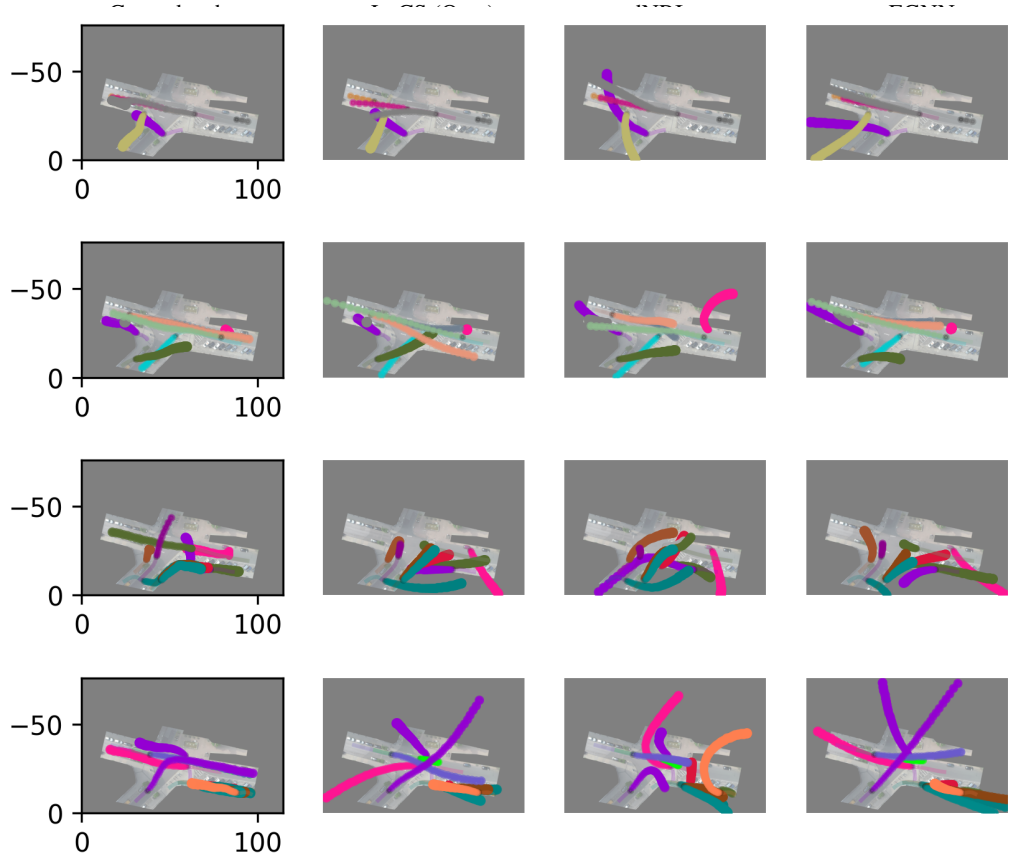


Figure 9: Qualitative results on inD

[61] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, 2019.

[62] Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.-Y., Johnson, J., and Gkioxari, G. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

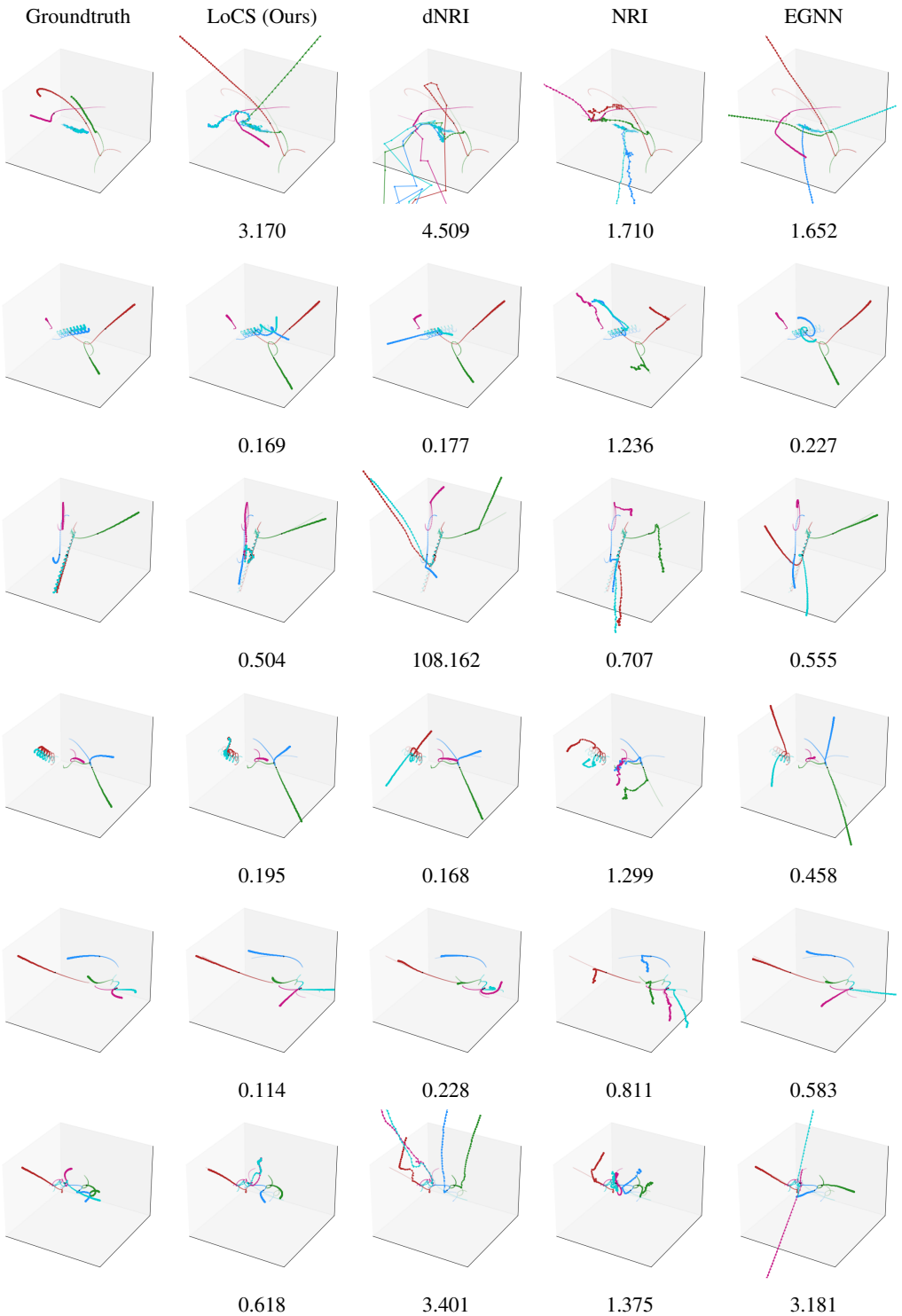


Figure 10: Qualitative results on interactive subset of charged particles, scenes #0 – #5

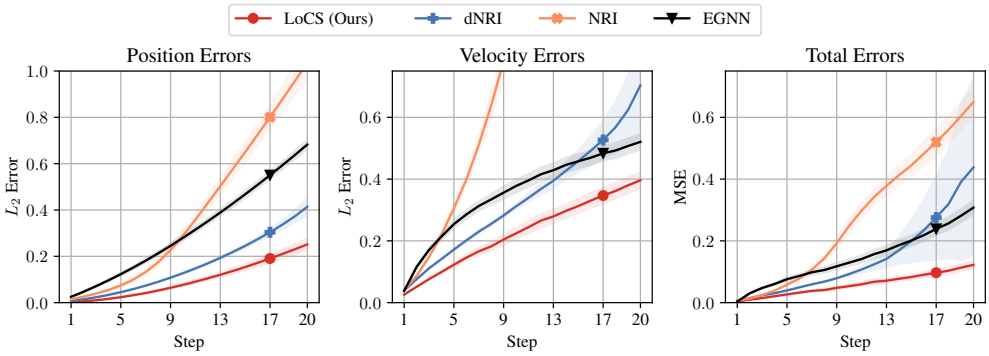


Figure 11: Results on Charged particles dataset

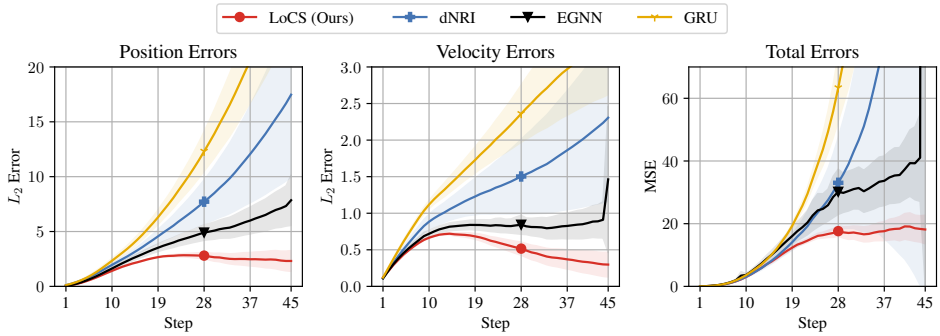


Figure 12: Results on InD dataset

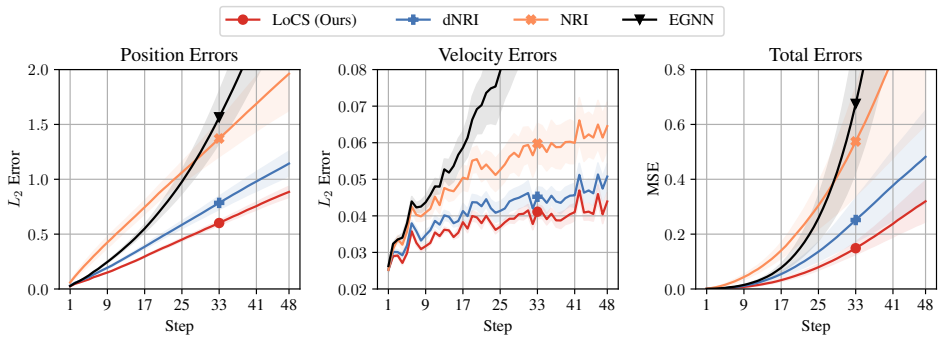


Figure 13: Results on motion capture (#35)

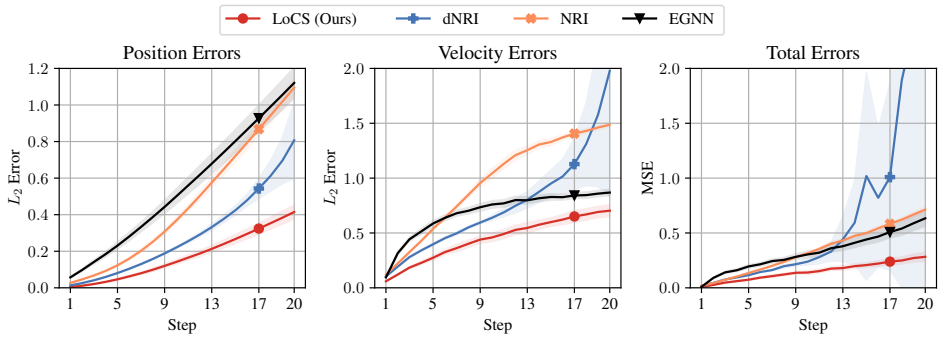


Figure 14: Results on Charged particles interactive subset

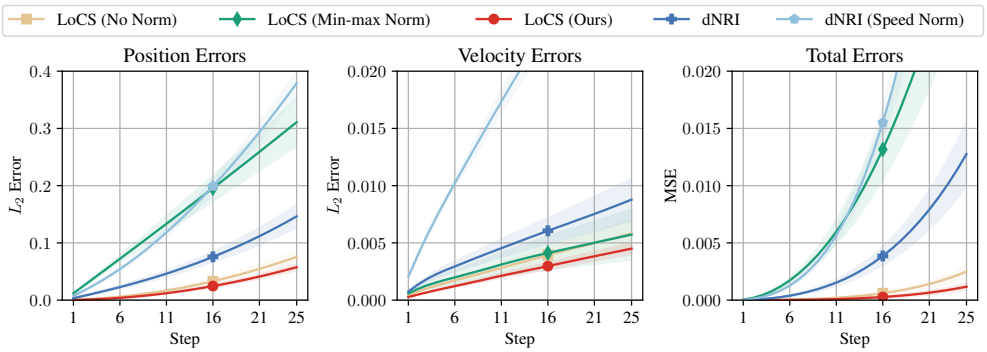


Figure 15: Results on synthetic dataset; impact of speed norm normalization

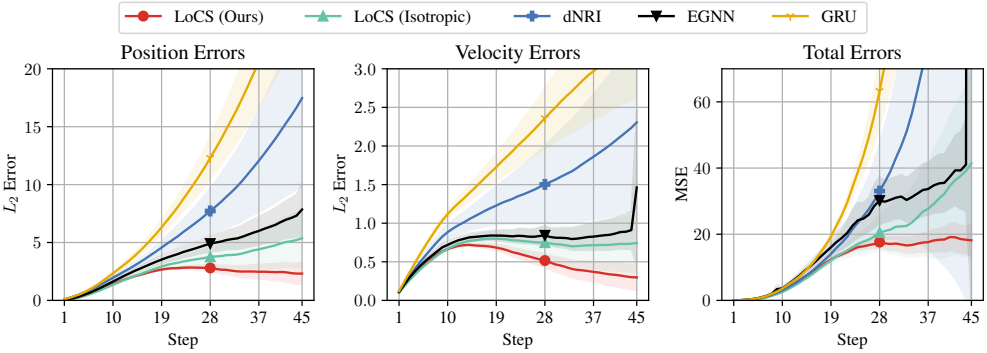


Figure 16: Results on InD dataset; impact of anisotropic filtering

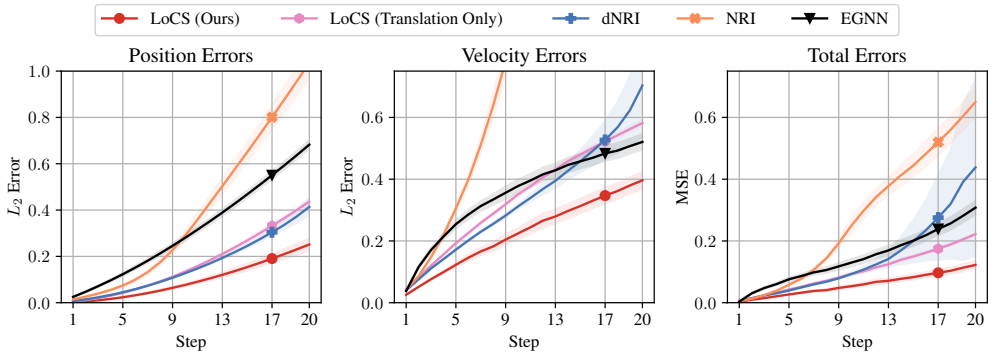


Figure 17: Results on charged particles dataset; impact of rotation in roto-translated local coordinate frames