Learning Neural Representation of Camera Pose with Matrix Representation of Pose Shift via View Synthesis

Yaxuan Zhu, Ruiqi Gao, Siyuan Huang, Song-Chun Zhu, and Ying Nian Wu
Department of Statistics, University of California, Los Angeles (UCLA)
{yaxuanzhu, ruiqigao, huangsiyuan}@ucla.edu, {sczhu, ywu}@stat.ucla.edu

Abstract

How to effectively represent camera pose is an essential problem in 3D computer vision, especially in tasks such as camera pose regression and novel view synthesis. Traditionally, 3D position of the camera is represented by Cartesian coordinate and the orientation is represented by Euler angle or quaternions. These representations are manually designed, which may not be the most effective representation for downstream tasks. In this work, we propose an approach to learn neural representations of camera poses and 3D scenes, coupled with neural representations of local camera movements. Specifically, the camera pose and 3D scene are represented as vectors and the local camera movement is represented as a matrix operating on the vector of the camera pose. We demonstrate that the camera movement can further be parametrized by a matrix Lie algebra that underlies a rotation system in the neural space. The vector representations are then concatenated and generate the posed 2D image through a decoder network. The model is learned from only posed 2D images and corresponding camera poses, without access to depths or shapes. We conduct extensive experiments on synthetic and real datasets. The results show that compared with other camera pose representations, our learned representation is more robust to noise in novel view synthesis and more effective in camera pose regression.

1. Introduction

With the advance of deep neural network (DNN), there has been a series of successful works that employ DNN in camera pose estimation [17, 16, 2, 26, 1, 20] or object pose estimation [5]. In contrast, novel view synthesis is in the opposite direction that maps the camera pose and 3D scene representation back to the posed 2D image under certain view [6, 30]. A fundamental problem in both lines of work is to find effective representations of the camera pose [39]. Existing methods include representing the agent's position in 3D Cartesian coordinate, and the 3D orientation can be represented by Euler angle, axis-angle, $SO(3)$ rotation matrices, quaternions or log quaternions. These representations are mainly defined in manually designed coordinates where each dimension has highly abstract semantics, which could be suboptimal when involved in the optimization with deep neural networks. It is desirable to have learning-based representations for camera poses.

Recently, [8] proposes a representational model of grid cells in the entorhinal cortex of mammalian brains. Grid cells have been found participating in mental self-navigation and they fire at strikingly regular hexagon grids of positions when the agent moves within an open field. The representational model in [8] consists of a vector representation of agent's self-position, coupled with a matrix representation of agent's self-motion. When the agent undergoes a certain self-motion in the 2D space, the vector of self-position is rotated by the matrix of self-motion on a 2D sub-manifold in the mental space. Such a model achieves self-navigation and learns hexagon grid patterns of grid cells, which has the promise to be biologically plausible.

Inspired by [8, 9], we propose an approach towards learning neural representation of camera pose, coupled with representation of local camera movement. Specifically, given 2D posed images of a 3D scene and their corresponding camera poses, we assume a shared vector representation for the underlying 3D scene and a distinct vector representation for the camera pose of each 2D image. When the camera has a local displacement, the vector of 3D scene remains unchanged while the vector of camera pose is rotated by the matrix representation of camera movement (Figure 1). We further parametrize the matrix representation by matrix Lie group and the corresponding matrix Lie algebra. The vector representations of camera poses and matrix presentations of camera movements can be shared across multiple scenes, so that they can be learned from multiple scenes to boost performance. The vectors of 3D scene and camera pose are concatenated together to generate the 2D image through a decoder network (Figure 2). The model is learned with only posed 2D images and camera poses, without extra knowl-

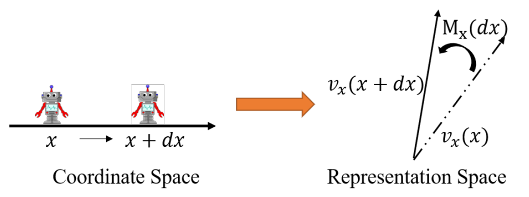


Figure 1: Illustration of our proposed pose representation. Take axis x as an example. The agent's position on axis x is mapped to a high dimensional vector and the agent's movement along axis x is modeled as a rotation of the vector.

edge such as depths or shapes. We perform various experiments on synthetic and real datasets in the context of novel view synthesis and camera pose regression.

The contributions of our work include:

1. We propose a method for learning neural camera pose representation coupled with neural camera movement representation.
2. We associate this representational model with the agent's visual input through a generative model.
3. We demonstrate that the learned neural representation is effective as the target representation in camera pose regression.

2. Related work

2.1. Representing camera orientation and position

The simplest way to represent orientation is by Euler angle. However, as [17, 16] point out, Euler angle wraps around at 2π and is not injective to 3D rotation, and thus can be difficult to learn. [1] uses $SO(3)$ rotation matrices to represent the relative orientation rotation between a pair of images. $SO(3)$ rotation matrices are an over-parameterized representation of rotation which has the property of orthonormality. However, it is in general difficult to enforce the orthonormality constraint when learning a $SO(3)$ representation through back-propagation. [34, 22] use axis-angle representation, which represents 3D orientation by the direction of axis of rotation as well as the magnitude of the rotation. Similar to Eluer angles, this representation also has the problem of repetition around the 2π radians. PoseNet and its variants [17, 16] propose to use quaternions. Quarernions, or more specfically, quaternions with unit length, are a 4-D continuous and smooth representation of rotation. MapNet [2] further proposes to use log quaternions to avoid over-parametrization. These quaternion-based methods achieve state-of-the-art results in the area of absolute camera pose regression. [39] argues that these representations are not continuous and proposes another 5D or 6D representation for orientation. All these representations are manually designed and pre-defined. In contrast, [9] proposes a neural representation of position and motion to explain the emergence of grid cell pattern. However, [9] only considers motion in 2D space and does not take visual input into consideration. Our method can be seen as a generalization of [9]. Our method models both position and orientation and their corresponding changes in 3D environments, and we associate position representations with visual inputs. The concept of position embedding is also used in other areas such as natural language processing. For example, transformer-based models such as BERT [4] or GPT [25] have a high dimensional embedding of the position of word in the sentence. These embeddings [35, 28, 10, 36] can be either learnable or predefined. We introduce learnable representations for camera pose in 3D vision. Our rotation loss enforces translation invariance, which serves as a regularization on the learned representations.

2.2. Novel view synthesis

Learning neural 3D scene representation is a fundamental problem in 3D vision, and a compelling way to evaluate the learned representations is by novel view synthesis. One line of work [30, 23, 33] incorporates prior knowledge of rendering such as rotation and projection to enforce the consistency between different views, such as NeRF [23]. Another theme [31, 37, 6] learns neural representations purely from the perception of the agent, without extra 3D prior knowledge. Our model belongs to the latter. Different from previous methods, we also learn neural representations of the camera pose and camera movement, and the representations of 3D scene and camera pose are disentangled in an unsupervised manner. [31, 37] infer the scene representation from a single image or a pair of images, while [6] assumes that the representation can be obtained from a small batch of images. Compared to these methods, our model is able to utilize posed images of various scenes to update the shared camera pose representations.

2.3. Interpretable representation

In generative modeling, learning interpretable latent representation is a long-standing target. Specifically, the goal is to learn latent vectors such that each dimension or subvector is aligned with an independent factor or concept. This can be done either with supervision [19, 24] or without supervision [11, 18, 15]. Besides vector representation, [21, 37, 14, 7] learns matrix representation of image transformation that operates on the latent vector representation.

Our model is a combination of both vector and matrix representations. On the one hand, we disentangle the vector representations of each individual scene and camera pose. On the other hand, we model the movement of the camera pose by matrix representation, which is in the form of matrix Lie group and matrix Lie algebra. In terms of parametrization of the matrix representation, [37] uses predefined and fixed rotation matrix, [21] learns a fixed matrix for each type of variation, and [14] parametrizes 2D ego-motion operated on 2D images. Different from these methods, we parametrize the matrix representation of camera movement as a nonlinear function of the movement in 3D that can take continuous values and operate on the vector representations of 3D scenes.

2.4. Deep pose regression models

Deep pose regression models [27] can be categorized into absolute camera pose regression (APR) [17, 16, 2] which directly predicts the camera pose given an input image, and relative camera pose estimation (RPR) [26, 1, 20] that predicts the pose of a test image relative to one or more training images. In this work, we adopt the APR setting while the method can also be easily adapted to the RPR setting. Note that our focus is to compare the effectiveness of different camera pose representations, which is orthogonal to the other methods that specifically target at improving the performance of pose regression.

3. Representational model

Suppose an agent move in a 3D environment with head rotations. There are at most 6 degrees of freedom (DOF), i.e., the position of the agent (x, y, z) and its head orientation (α, β, γ). We denote them as the pose of the agent $\boldsymbol{p} = (x, y, z, \alpha, \beta, \gamma)$. Following the idea of [9], we encode each DOF by a d-dimensional sub-vector $\boldsymbol{v}_l(l), l \in \{x, y, z, \alpha, \beta, \gamma\}$. From the embedding point of view, essentially we embed the 1D domain in $\mathbb{R}^1$ as a 1D manifold in a higher dimensional space $\mathbb{R}^d$. We limit each sub-vector to have unit length, i.e., we further assume the 1D manifold to be a circle. For notation simplicity, we concatenate those sub-vectors to a pose vector $\boldsymbol{v}(\boldsymbol{p})$. When the camera makes a movement $\delta\boldsymbol{p} = (\delta x, \delta y, \delta z, \delta\alpha, \delta\beta, \delta\gamma)$, the camera pose changes from $\boldsymbol{v}(\boldsymbol{p})$ to $\boldsymbol{v}(\boldsymbol{p} + \delta\boldsymbol{p})$. See Figure 1 for an illustration of our proposed framework.

3.1. Modeling movement as vector rotation

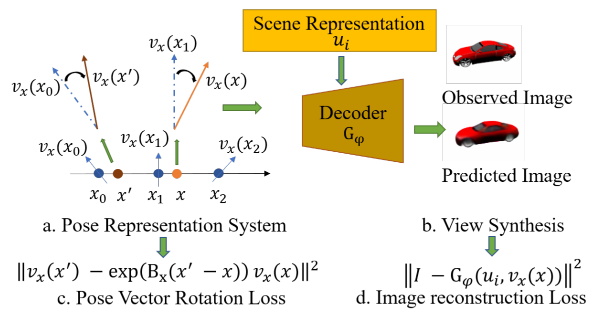
We start from considering an infinitesimal camera movement $\delta\boldsymbol{p}$. For each DOF $l \in \{x, y, z, \alpha, \beta, \gamma\}$, we propose the following model:

$$\boldsymbol{v}_l(l + \delta l) = \boldsymbol{M}_l(\delta l)\boldsymbol{v}_l(l) + \boldsymbol{o}(\delta l), \tag{1}$$

where $\boldsymbol{M}_l(\delta l)$ is a $d \times d$ matrix depending on δl. Given that δl is infinitesimal, the model can be further parametrized as

$$\boldsymbol{v}_l(l + \delta l) = (\boldsymbol{I} + \boldsymbol{B}_l\delta l)\boldsymbol{v}_l(l) + \boldsymbol{o}(\delta l), \tag{2}$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{B}_l$ is a $d \times d$ matrix that needs to be learned. We assume $\boldsymbol{B}_l$ to be skew-symmetric i,e. $\boldsymbol{B}_l = -\boldsymbol{B}_l^T$. This assumption guarantees



Scene Representation u_i
Decoder G_φ
Observed Image
Predicted Image
$v_x(x_1)$
$v_x(x_0)$
$v_x(x')$
$v_x(x)$
$v_x(x_0)$
$v_x(x_1)$
$v_x(x_2)$
x_0 x' x_1 x x_2
a. Pose Representation System
b. View Synthesis
$\|v_x(x') - \exp(B_x(x' - x))\, v_x(x)\|^2$
c. Pose Vector Rotation Loss
$\|I - G_\varphi(u_i, v_x(x))\|^2$
d. Image reconstruction Loss

Figure 2: Illustration of our framework. (a) Pose vector for a given position x is obtained by rotating its nearby grid vector. (b) Pose vector is fed-in a decoder together with a scene representation vector to predict image under certain view. (c) The rotation consistency of our pose representation system is enforced through pose rotation loss. (d) The decoding ability of our pose representation system is enforced through image reconstruction loss.

that $(\boldsymbol{I} + \boldsymbol{B}_l\delta l)(\boldsymbol{I} + \boldsymbol{B}_l\delta l)^T = \boldsymbol{I} + \boldsymbol{o}(\delta l^2)$, i.e., $(\boldsymbol{I} + \boldsymbol{B}_l\delta l)$ is approximately an orthogonal matrix. From the geometric perspective, it maps the movement along l axis in 1D space to rotation of the vector in the high-dimensional latent space. In practice, we only need to parametrize the upper triangle of $\boldsymbol{B}_l$ as trainable parameters and the lower triangle of $\boldsymbol{B}_l$ is taken to be the negative of the upper triangle. We further assume $\boldsymbol{B}_l$ to be block-diagonal so that the total number of parameters can be greatly reduced. If there are movements on multiple DOFs, we only need to rotate each sub-vector of DOF independently.

As pointed out by [9], equations 1 and 2 can be justified as a minimally simple recurrent model. To model the movement in the latent space, the most general form is $\boldsymbol{v}_l(l + \delta l) = F(\boldsymbol{v}_l(l), \delta l)$, i.e., the pose vector for the new pose is a function of the one for the old pose and the movement. Given that δl is infinitesimal, we can apply the first-order Taylor expansion: $\boldsymbol{v}_l(l + \delta l) = \boldsymbol{v}_l(l) + f(\boldsymbol{v}_l(l))\delta l + \boldsymbol{o}(\delta l)$, where we use $f(\boldsymbol{v}_l(l))$ to denote the first derivative of $F(\boldsymbol{v}_l(l), \delta l)$, i.e., $f(\boldsymbol{v}_l(l)) = \frac{\partial}{\partial \delta l} F(\boldsymbol{v}_l(l), \delta l)|_{\delta l = 0}$. When the movement $\delta l = 0$, we should have that $F(\boldsymbol{v}_l(l), 0) = \boldsymbol{v}_l(l)$. Then a minimally simple model is to assume $f(\boldsymbol{v}_l(l))$ as a linear transformation , i.e. $f(\boldsymbol{v}_l(l)) = \boldsymbol{B}_l\boldsymbol{v}_l(l)$, and $\boldsymbol{v}_l(l + \delta l) = \boldsymbol{v}_l(l) + \boldsymbol{B}_l\boldsymbol{v}_l(l)\delta l + o(\delta l)$. As we will discuss in 3.3, for finite movement $\Delta\boldsymbol{p}$, we recurrently apply the model of infinitesimal $\delta\boldsymbol{p}$, so that the matrix representation becomes a matrix Lie group.

3.2. Polar system for position change in 2D

If the movement of the agent is constrained in a 2D environment, we follow [9] to use a polar coordinate system to model the change of position, which corresponds to the egocentric perspective and could be potentially more biological plausible. Specifically, let $\boldsymbol{x} = (x, y)$ be the position of the agent in the 2D space, instead of using individual vector $\boldsymbol{v}_x$ and $\boldsymbol{v}_y$ to represent the position, we represent position in a single vector $\boldsymbol{v}_{\boldsymbol{x}}(\boldsymbol{x})$ and the movement is captured by direction θ and distance δr. We have $\delta \boldsymbol{x} = (\delta x, \delta y) = (\delta r \cos\theta, \delta r \sin\theta)$. The representational model under this polar coordinate system is:

$$\boldsymbol{v}_{\boldsymbol{x}}(\boldsymbol{x} + \delta\boldsymbol{x}) = (\boldsymbol{I} + \boldsymbol{B}(\theta)\delta r)\boldsymbol{v}_{\boldsymbol{x}}(\boldsymbol{x}) + \boldsymbol{o}(\delta r). \quad (3)$$

The $\boldsymbol{B}(\theta)$ is a function of θ and is skew-symmetric. $\boldsymbol{B}(\theta)$ models the change of position along the direction θ. If the agent changes the direction of movement from θ to $\theta + \delta\theta$, then we assume

$$\boldsymbol{B}(\theta + \delta\theta) = (\boldsymbol{I} + \boldsymbol{C}\delta\theta)\boldsymbol{B}(\theta) + \boldsymbol{o}(\delta\theta), \quad (4)$$

where $\boldsymbol{C}$ is another skew-symmetric matrix to learn. The geometric interpretation is that if the agent changes direction, $\boldsymbol{B}(\theta)$ is rotated by another matrix $\boldsymbol{C}$.

For camera movement in 3D environment, such coupled representation in polar coordinate will end up with too many matrix representations to learn. Therefore, we restrict ourselves in using it only in 2D space, and use the vector-matrix representations that are disentangled for each DOF as proposed in 3.1 for general 3D movements.

3.3. Matrix Lie group for finite movement

So far we have discussed the formulation for infinitesimal movements above. In this subsection we generalize to finite movements. Suppose the agent has a finite movement Δl along the axis $l \in \{x, y, z, \alpha, \beta, \gamma\}$. We can divide this movement into N steps, so that as $N \to \infty$, $\frac{\Delta l}{N} \to 0$, and

$$\begin{aligned}\boldsymbol{v}_l(l + \Delta l) &= (\boldsymbol{I} + \boldsymbol{B}_l(\frac{\Delta l}{N}) + \boldsymbol{o}(\frac{1}{N}))^N \boldsymbol{v}_l(l) \\ &\to \exp(\boldsymbol{B}_l \Delta l)\boldsymbol{v}_{\boldsymbol{l}}(l). \quad (5)\end{aligned}$$

This underlies the relationship between matrix Lie algebra and matrix Lie group. Specifically, the set of $\boldsymbol{M}_l(\Delta l) = \exp(\boldsymbol{B}_l \Delta l)$ for $\Delta l \in \mathbb{R}$ forms a matrix Lie group. The tangent space of $\boldsymbol{M}(\Delta l)$ at identity is the corresponding matrix Lie algebra. $\boldsymbol{B}_l$ is the basis of this tangent space, and is also called as the generator matrix.

For a finite but small Δl, $\exp(\boldsymbol{B}_l \Delta l)$ can be approximated by a second-order Taylor expansion

$$\exp(\boldsymbol{B}_l \Delta l) = \boldsymbol{I} + \boldsymbol{B}_l \Delta l + \frac{1}{2}\boldsymbol{B}_l^2 \Delta l^2 + \boldsymbol{o}(\Delta l^2). \quad (6)$$

For a large finite change in each axis, we can divide it into a series of small finite changes, expand each change using second-order Taylor expansion and multiply them together.

3.4. Theoretical understanding of our model

A deep theoretical result from mathematics, namely the Peter-Weyl theorem [32], inspires our work. It says that for a compact Lie group, if we can find an irreducible unitary representation, i.e., each element $\boldsymbol{x}$ of the group is represented by a unitary (or orthogonal) matrix $\boldsymbol{M}(\boldsymbol{x})$, then the matrix elements $(M_{ij}(\boldsymbol{x}))$ form a set of orthogonal basis functions for the general functions of $\boldsymbol{x}$. This is a deep generalization of Fourier analysis. In our case, the learned vector representation $\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{M}(\boldsymbol{x})\boldsymbol{v}(0)$ are linear compositions of the above basis functions, and the elements $(v_i(\boldsymbol{x}))$ serve as a more compact set of basis functions for representing general functions of $\boldsymbol{x}$. Our method can be used to represent the pose of the camera and objects in general. The continuous changes of the pose in the physical space generally form a Lie group. Our learned vector and matrix system forms a representation of the pose and its change in the neural space.

3.5. Implementation of pose representation

Suppose we want to learn the representation of axis l, whose value ranges in $[a, b]$. For orientation, the angles is of range [0°, 360°), while for position, we can predefine the largest range the agent can move within. We divide this range into multiple grids and we learn an individual vector at each grid point. Given an arbitrary position $l \in [a, b]$, we first find its nearest grid point and the corresponding vector representation, and then we rotate this vector to the target position by the matrix representation depending on the distance between this nearest grid position and the target position. See Figure 2. Since we can set the length of grid to be relatively small, the distance between the grid and target positions is also small, so that we can use second-order Taylor expansion in Equation 6 to approximate the matrix representation.

3.6. Decoding to posed 2D images

To associate the camera pose representation with visual input, more specifically the posed 2D images, we propose a decoder or emission model. For each 3D scene, suppose we are given multiple posed 2D images $\mathbf{I}$ and the corresponding camera poses $\boldsymbol{p}$. Then we assume a shared vector representation $\boldsymbol{u}$ of the 3D scene, and obtain the vector representation of the camera pose $\boldsymbol{v}(\boldsymbol{p})$ as described in 3.5. We learn a decoder G_ϕ that maps $\boldsymbol{u}$ and $\boldsymbol{v}(\boldsymbol{p})$ to the image space to reconstruct $\mathbf{I}$

$$\hat{\mathbf{I}} = G_\phi(\boldsymbol{u}, \boldsymbol{v}(\boldsymbol{p})), \quad (7)$$

where ϕ denotes parameters in the decoder network.

4. Learning and inference

4.1. Learning through view synthesis

For a general 3D environment, the unknown parameters of the proposed model include (1) $\boldsymbol{v}(\boldsymbol{p})$ for any $\boldsymbol{p}$ on grid positions, (2) $\boldsymbol{B}_l$ for any $l \in \{x, y, z, \alpha, \beta, \gamma\}$, and (3) parameters ϕ in G_ϕ. To learn these parameters, we define a loss function $L = \lambda_1 L_{\text{rec}} + \lambda_2 \sum_{l \in \{x,y,z,\alpha,\beta,\gamma\}} L_{\text{rot},l}$, where

$$\begin{aligned} L_{\text{rec}} &= \mathbb{E}_{\mathbf{I}} \left\| \mathbf{I} - G_\phi(\boldsymbol{u}, \boldsymbol{v}(\boldsymbol{p})) \right\|^2, \\ L_{\text{rot},l} &= \mathbb{E}_{l,\Delta l} \left\| \boldsymbol{v}_l(l+\Delta l) - \exp(\boldsymbol{B}_l(\Delta l))\boldsymbol{v}_l(l) \right\|^2. \end{aligned} \tag{8}$$

L_{rec} is the reconstruction loss for view synthesis, which enforces the decoding of the pose and scene representations to reconstruct the observation. The expectation is estimated by Monte Carlo samples. L_{rot} stands for rotation loss, which serves to constrain $\boldsymbol{v}_l$ so that the learned pose representations of different poses can be transformed to each other based on our representational model (Equation 5). The expectation term in L_{rot} can be approximated by randomly sampled pairs of poses $\boldsymbol{p}$ and $\boldsymbol{p}+\Delta\boldsymbol{p}$ that are relatively close to each other, which means that we have infinite amount of data for this loss term.

If the movement of camera pose is in a 2D space and we employ the polar coordinate system, then part (2) of the unknown parameters becomes $\boldsymbol{B}_l$ for any $l \in \{\alpha, \beta, \gamma\}$, $\boldsymbol{B}(\theta)$ and C. The loss functioin is defined as $L = \lambda_1 L_{\text{rec}} + \lambda_2 \sum_{l \in \{\alpha,\beta,\gamma\}} L_{\text{rot},l} + \lambda_3 L_{\text{rot},\boldsymbol{x}} + \lambda_4 L_{\text{rot},\theta}$, where L_{rec} and $L_{\text{rot},l}$ follow equation 8 and

$$\begin{aligned} L_{\text{rot},\boldsymbol{x}} &= \mathbb{E}_{\boldsymbol{x},\Delta\boldsymbol{x}} \left\| \boldsymbol{v}_{\boldsymbol{x}}(\boldsymbol{x}+\Delta\boldsymbol{x}) - \exp(\boldsymbol{B}(\theta)\Delta r)\boldsymbol{v}_{\boldsymbol{x}}(\boldsymbol{x}) \right\|^2, \\ L_{\text{rot},\theta} &= \mathbb{E}_{\boldsymbol{x}} \left\| \boldsymbol{B}(\theta+\Delta\theta) - \exp(\boldsymbol{C}\Delta\theta)\boldsymbol{B}(\theta) \right\|^2. \end{aligned} \tag{9}$$

For training, we minimize L by iteratively updating the decoder G_ϕ (as well as our scene representation $\boldsymbol{u}$) and our pose representation system v_l, M_l for $l \in \{x, y, z, \alpha, \beta, \gamma\}$. In practice, the decoder is parameterized by a multi-layer deconvolutional neural network. Besides the latent vector on top of the decoder, we also learn a scene-dependent vector at each following layers using AdaIN [12]. We normalize the scene vector at the top layer of the decoder to have unit norm so that it has the same magnitude as the pose representation. We find this helps optimization. More details can be found in Supplementary.

4.2. Inference by pose regression

With the learned pose representation, we can then use it as the target output for camera pose regression. Specifically, for each DOF, we train a separate inference network $E_{\xi l}$ that maps the observed posed 2D image $\mathbf{I}$ to the pose representation $\boldsymbol{v}_l(l)$. The loss function is defined as the L_2 distance between the inferred and learned pose presentations

$$L_l = \mathbb{E}_{\mathbf{I}} \left\| \boldsymbol{v}_l(l) - E_{\xi l}(\mathbf{I}) \right\|^2. \tag{10}$$

In practice, $E_{\xi l}$ is parameterized by a convolutional neural network where ξ denotes the parameters and we introduce some scene-dependent parameters using AdaIN. For different DOFs, the inference networks share common lower layers but with different top fully-connected layers.

For testing, given an unseen posed image $\mathbf{I}$, we can get the inferred pose representation $\hat{\boldsymbol{v}}_{\boldsymbol{l}}$ from our inference model, and decode the predicted pose by:

$$\hat{l} = \arg\min_l \left\| \boldsymbol{v}_l(l) - \hat{\boldsymbol{v}}_{\boldsymbol{l}} \right\|^2, \; l \in (x, y, z, \alpha, \beta, \gamma) \tag{11}$$

5. Experiments

In this section, we demonstrate the efficacy of our learned pose representations in both view synthesis and pose regression tasks. For view synthesis, we mainly compare with the Generative Query Network(GQN) [6]. For pose regression, we compare our learned neural representations of camera pose with other commonly used pose representations, including the Euler angle, the sinusoidal representation used in GQN, and the quaternions (as well as log quaternions) representations used in the PoseNet [17, 16] and MapNet [2], by evaluating the pose estimation accuracy. More details of implementation can be found in Supplementary. Our code and pretrained models can be found at https://github.com/AlvinZhuyx/camera_pose_representation.

5.1. Datasets

GQN rooms. GQN [6] introduces a synthetic dataset with 2 million synthetic scenes, where each scene contains various objects, textures, and walls. The agent can navigate in a 2D space and rotate the head horizontally in the scenes. Each scene contains 10 rendered 64×64 RGB images. We use the version of the dataset where the camera moves freely and the objects do not rotate around their axes. We sample $200,000$ scenes from the dataset. For each scene, we sample 9 images for training and use the left one image for testing. Since this dataset contains a large number of simple scenes with a small number of images for each scene, instead of learning an individual scene representation vector for each scene, we learn an encoder to encode the scene representation online similar to [6]. Since the agent has 2 DOFs for position and 1 DOF for orientation, our pose vector contains one position sub-vector in the polar coordinate system and one orientation sub-vector. Each sub-vector has 96 dimensions. We assume that B is block-diagonal with six blocks, and each block is 16×16.

ShapeNet v2. We use the images generated by [30] from the *car* category of ShapeNet v2 dataset [3]. This dataset contains 2,151 object models. For each scene, the instance locates at the center of a sphere. The virtual agent can move on the surface of this sphere, with its camera pointing to the center. Therefore, the agent has 2 DOFs, and we use 2 orientation angles to denote its position on the sphere. Each instance contains 500 different views of 128×128 rendered RGB images, where we randomly sample 100 images for training and leave the others for testing. The pose representation contains two sub-vectors of two orientation angles. Each sub-vector has a dimension of 96, and B has six 16×16 blocks. We learn an individual scene representation vector $\boldsymbol{u}$ for each instance.

Gibson Environment. The Gibson Environment [38] provides tools for rendering images corresponding to different views in a room, which we use to generate a synthetic dataset. We refer to this dataset as Gibson rooms. Specifically, we select 20 areas of size 2m $\times$ 2m from different rooms. For each area, we randomly render about 28k 128×128 RGB images of different views. We fix the camera height and constrain the camera to rotate only horizontally. Compared to GQN rooms and ShapeNet car, this synthetic dataset contains more realistic and complicated indoor scenes, which could be more challenging. Moreover, it includes fewer scenes while for each scene, images from abundant views are provided. Therefore, incorporating view-based information becomes very important. The agent has 2 DOFs for position and 1 DOF for orientation, which corresponds to a position sub-vector in the polar coordinate system and one orientation sub-vector. The dimensions of the sub-vectors and B are the same as the ones for GQN rooms dataset.

7 Scenes Dataset. Microsoft 7 Scenes [29] is a widely used dataset for camera pose estimation. It contains RGB-D images for seven different indoor scenes. Each scene has several trajectories for training and testing. In our experiment, we follow the training and testing split in [29], and we only use RGB images without depth information. We translate and align the position coordinates of scenes and ensure that all the trajectories locate in a 4m $\times$ 1.5m $\times$ 3m cuboid. The agent has 6 DOFs, so the pose representation vector contains 6 sub-vectors. We assume that each sub-vector has a dimension of 32, and each B has four 8×8 blocks. We mainly use this dataset for camera pose regression. We resize the images to 128×128 when training the decoder and pose representation system. We use shared pose representations for all the seven scenes and distinct scene representation for each of them. When performing pose regression, following [16, 2], we train an individual inference model for each scene and resize the input images so that the shortest side is of length 256.

5.2. Novel view synthesis

The first question is whether our learned pose representation is meaningful. We answer this by testing our learned representations on novel view synthesis task. The experimental results demonstrate that our learned representations can generate a novel view of a scene of high quality. Figure 3 shows the qualitative results, and Figure 4 shows the quantitative results in terms of Peak Signal-to-Noise Ratio (PSNR). We compare the results with GQN. For GQN, we use the implementation by [13] and the same training and testing splits as ours. We use 8 generation layers and set the shared core option to be False. We add extra convolution and de-convolution layers when dealing with images of size 128×128. The total number of parameters for this GQN implementation is 114M. In contrast, our model only has less than 9M parameters.

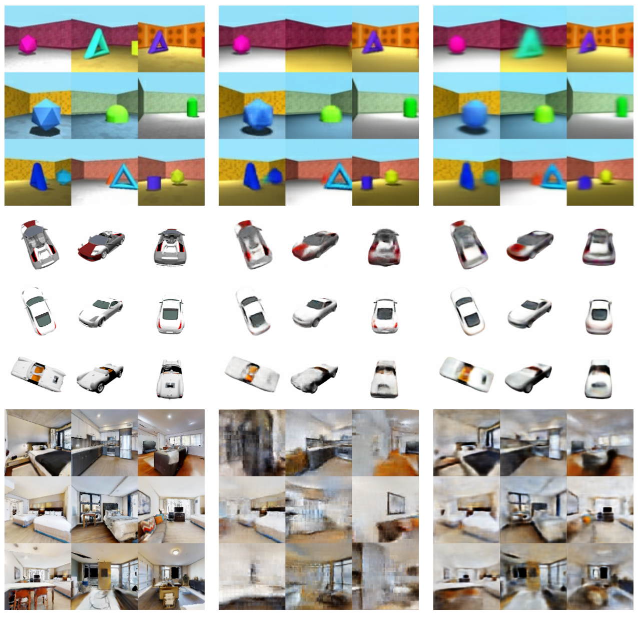
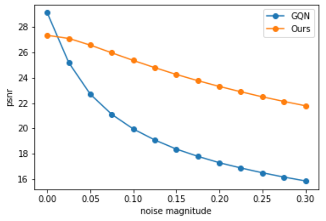


Figure 3: Qualitative results for novel view synthesis. *Top*: GQN rooms. *Middle*: ShapeNet car. *Bottom*: Gibson rooms.

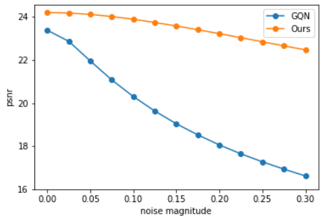
From Figure 3 and Figure 4 (noise magnitude of 0.0 corresponds to novel view synthesis test result), we see that for GQN rooms dataset, our model gets a bit worse but comparable results with the GQN model. For ShapeNet car dataset, which contains complex instances, our model generates more consistent and clearer results compared with GQN. For Gibson rooms dataset, which is more complicated, GQN fails to capture the relationship. The reconstruction only captures some specific views and does not generalize to other views. On the other hand, our learned model is able to generate a query view corresponding to our pose representation. This is probably because that the 3D scene representations in our method are learned by all the 2D posed images of the scene.

5.3. Robustness to pose noise

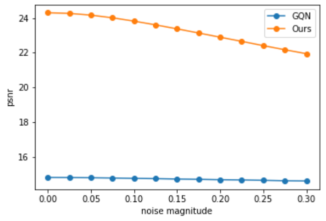
Next, we try to answer why we need that representation and what is the advantage of such neural representation over directly using 6 DOFs coordinate representation in terms of novel view synthesis. One critical supporting evidence is that our learned neural pose representation is more robust to noise. Specifically, Figure 4 shows the changes of PSNR for our model versus the GQN model when some Gaussian



(a) GQN rooms



(b) ShapeNet car



(c) Gibson rooms

Figure 4: Quantitative results for novel view synthesis given different noise magnitudes. In each figure, we plot the PSNR over different magnitudes of noise introduced to the position vector. For a given noise magnitude α, if the i-th element in the position vector has a standard deviation β_i, then we add a Gaussian Noise $N(0, (\alpha\beta_i)^2)$ to the corresponding element. Noise magnitude 0.0 corresponds to the novel view synthesis test result. We compare with GQN on three datasets.

Representations	ShapeNet car	GQN rooms			Gibson rooms		
	orientation	x	y	orientation	x	y	orientation
(x, y, α)	7.75°	0.069	0.071	12.00°	0.043m	0.041m	7.03°
$(x$, y, axis-angle)	11.29°	-	-	-	-	-	-
$(x$, y, $\sin(\alpha)$, $\cos(\alpha)))$	7.29°	0.108	0.104	16.46°	0.033m	0.034m	1.19°
$(x$, y, $q)$	4.28°	**0.050**	**0.048**	5.34°	0.043m	0.042m	1.21°
$(x$, y, $\log q)$	5.35°	0.051	0.051	7.44°	0.028m	0.027m	1.17°
ours	**2.85°**	0.053	0.053	**4.07°**	**0.021m**	**0.020m**	**0.87°**

Table 1: Camera pose estimation errors on different datasets. We compare with several camera pose representations. $(x$, y, $\alpha)$ denotes the representation that uses x, y, z coordinate to represent position and Euler angle to represent orientation. $(x$, y, axis-angle) denotes using axis-angle representation for rotation. Note that for GQN rooms and Gibson rooms datasets, the agent only has one DOF of rotation. Therefore, the axis-angle representation degrades to one Euler angle representation, and its results should be the same as the Euler angle. $(x$, y, $\sin(\alpha), \cos(\alpha))$ denotes using sinusoidal functions to represent orientation. $(x$, y, $q)$ denotes the unit quaternions representation used in [16] while $(x$, y, $\log q)$ stands for the logarithm quaternions representation proposed in [2]. Our method uses learned pose vectors for both camera position and orientation. We report the average prediction error for each dataset. For ShapeNet car dataset, the camera is located on a sphere, so we only need to predict the orientation angle. For GQN rooms and Gibson rooms datasets, we predict both the camera position and orientation. For GQN rooms, the range of each scene is from -1.0 to 1.0. For Gibson rooms, we render each scene to an area of 2m $\times$ 2m.

noise with various magnitudes is added to the pose representations. We observe that the performance of the GQN model degrades quickly as the magnitude of added noise increases. This is not surprising since GQN directly uses coordinate representation for position and orientation and thus is vulnerable to noise interference. On the other hand, our learned representation embeds the camera pose to high dimensional space and is further regulated by the rotation loss, and thus is more robust to noise.

5.4. Inference results

We further demonstrate that our learned representation is efficient to serve as the target output of pose regression. In the camera pose regression task, the camera position is usually represented using 3D coordinate (x, y, z) and the camera orientation can be represented by various methods. The most straightforward one is to use the Euler angle to represent the orientation. Another representation is axis-angle representation. In [6], the authors use $(\sin(\alpha), \cos(\alpha))$ to represent each orientation angle. Besides, unit quaternions and logarithm of the unit quaternions are another two popular representations used in pose regression [17, 16, 2]. Comparing with those methods, we used learned neural representations for both camera position and orientation. We conduct the pose regression experiments on all four datasets we mentioned above. For our representation, Euler Angle representation and $(\sin(\alpha), \cos(\alpha))$ representation we use mean square error loss for regression. On 7 Scenes dataset, we also use L_1 norm loss for our representation. For quaternions and log quaternions representations, as suggested by [2], we use L_1 norm loss. For axis-angle representation, we find that for ShapeNet car dataset, using L_1 norm loss leads to better results. For Gibson rooms and GQN rooms datasets, since the agent can only rotate its head horizon-

Scene	PoseNet17[16]	PoseNet + log q[2]	PoseNet + log q (*)	ours
Chess	0.13m, 4.48°	**0.11m**, **4.29°**	0.17m 4.96°	0.12m 4.83°
Fire	**0.27m**, 11.30°	**0.27m**, 12.13°	0.36m 11.22°	**0.27m** **8.91°**
Heads	0.17m, 13.00°	0.19m, **12.15°**	0.20m 13.35°	**0.16m** 12.84°
Office	**0.19m**, **5.55°**	**0.19m**, 6.35°	0.23m 7.05°	**0.19m** 6.64°
Pumpkin	0.26m, **4.75°**	0.22m, 5.05°	0.26m 5.87°	**0.22m** 5.45°
Red Kitchen	**0.23m**, 5.35°	0.25m, **5.27°**	0.29m 6.10°	0.24m 6.10°
Stairs	0.35m, 12.40°	0.30m,11.29°	0.36m **10.18°**	**0.29m** 10.70°
Average	0.23m, 8.12°	0.22m 8.07°	0.27m 8.39°	**0.21m** **7.92°**

Table 2: Camera pose estimation errors on 7scenes dataset. We compare our results with PoseNet using quaternions (PoseNet17) and log quaternions (PoseNet + log q). The column PoseNet + log q(*) are the results we get by running the code provided by [2]. In the last column, we show the results using our learned pose representation. Following the convention, we report the median prediction errors here.

tally, the axis-angle representation degrades to a single Euler angle. For the two quaternions-related baselines, we employ the automatic weight tuning method proposed in [16] to make a fair comparison. Note that the main focus of this work is to compare different pose representations, and thus we do not include other improvement techniques (*e.g.*, including unlabeled data or relative pose loss between image pairs), as we consider them as orthogonal directions to improving the pose representations. More details can be found in Supplementary.

We first show the comparison results on GQN rooms, ShapeNet car, and Gibson rooms datasets in Table 1. For a fair comparison, we keep the same network structure for all the representations on each dataset and only change the final output layer. Since the dimension of our learned representation is higher than all the baseline representations, for a fair comparison, we add another fully-connected layer to these baseline inference networks so that the inference models have roughly the same number of parameters across different pose representations. According to Table 1, our representation consistently outperforms all the other representations, especially for orientation regression. For most configurations, our representation yields the best results in both orientation and position prediction. On GQN dataset, the quaternions and log quaternions representation achieve slightly better results in position prediction. However, their orientation prediction results are much worse than ours. A possible explanation is that we embed both the camera position and orientation as neural representations, and thus they are more consistent with each other. Besides, representing the rotation angles on a hyper-sphere in a high dimensional space may also make it easier for the model to regress.

We further compare our learned pose representations with the popular quaternions and log quaternions representations on 7 Scenes dataset using PoseNet. Following [2], we use a pre-trained ResNet34 as our feature extractor and 6 parallel fully-connected (FC) layers to predict the 6 pose sub-vectors. We employ color jittering as data augmentation and remove the dropout in the FC layers. The results are shown in Table 2. We compare our results with [16, 2]. We also run the code provided by [2] to re-train their model and report the results. The difference between the reported values and the reproduced results is probably due to the randomness and different versions of software [1]. Following the convention on this dataset, we report the median errors of location and orientation predictions. The result shows that, on average, our model outperforms all the baselines.

6. Conclusion and Future Work

We propose a framework for learning neural vector representations for both camera poses and 3D scenes, coupled with neural matrix representation for camera movements. The model is learned through novel view synthesis and can be used for camera pose regression. Our learned representation proves to be more robust against pose noise in the novel view synthesis task and works well as the estimation target for camera pose regression. We hope that our work can motivate further interest and study on learning neural representations for camera poses and joint representations for camera poses and 3D scenes. An interesting future direction is how to combine our method with the recent work of NeRF [23], which uses sinusoidal functions of very high frequencies. Our model can be adapted to this new generative model structure and may be able to learn more flexible camera pose representation.

Acknowledgment

The work is supported by NSF DMS-2015577; DARPA XAI N66001-17-2-4029; ARO W911NF1810296; ONR MURI N00014-16-1-2007.

[1]The code of [2] is originally implemented in python 2.7 and PyTorch 0.4.0 while we make minor adaptation to enable it to run in python 3.6 and PyTorch 1.2.0

References

[1] Vassileios Balntas, Shuda Li, and Victor Prisacariu. Relocnet: Continuous metric learning relocalisation using neural nets. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 751–767, 2018. 1, 2, 3

[2] Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Geometry-aware learning of maps for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2018. 1, 2, 3, 5, 6, 7, 8

[3] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 5

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 2

[5] Thanh-Toan Do, Ming Cai, Trung Pham, and Ian Reid. Deep-6dpose: Recovering 6d object pose from a single rgb image. *arXiv preprint arXiv:1802.10367*, 2018. 1

[6] SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018. 1, 2, 5, 7

[7] Ruiqi Gao, Jianwen Xie, Siyuan Huang, Yufan Ren, Song-Chun Zhu, and Ying Nian Wu. Learning vector representation of local content and matrix representation of local motion, with implications for v1. *arXiv preprint arXiv:1902.03871*, 2019. 2

[8] Ruiqi Gao, Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. Learning grid cells as vector representation of self-position coupled with matrix representation of self-motion. *arXiv preprint arXiv:1810.05597*, 2018. 1

[9] Ruiqi Gao, Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. A representational model of grid cells based on matrix lie algebras. *arXiv preprint arXiv:2006.10259*, 2020. 1, 2, 3

[10] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR, 2017. 2

[11] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016. 2

[12] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017. 5

[13] iShohei220. Pytorch implementation of generative query network. https://github.com/iShohei220/torch-gqn, Dec. 2018. 6

[14] Dinesh Jayaraman and Kristen Grauman. Learning image representations tied to ego-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1413–1421, 2015. 2, 3

[15] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 2

[16] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5974–5983, 2017. 1, 2, 3, 5, 6, 7, 8

[17] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015. 1, 2, 3, 5, 7

[18] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018. 2

[19] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In *Advances in neural information processing systems*, pages 2539–2547, 2015. 2

[20] Zakaria Laskar, Iaroslav Melekhov, Surya Kalia, and Juho Kannala. Camera relocalization by computing pairwise relative poses using convolutional neural network. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 929–938, 2017. 1, 3

[21] Or Litany, Ari Morcos, Srinath Sridhar, Leonidas Guibas, and Judy Hoffman. Representation learning through latent canonicalizations. *arXiv preprint arXiv:2002.11829*, 2020. 2, 3

[22] Siddharth Mahendran, Haider Ali, and René Vidal. 3d pose regression using convolutional neural networks. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2174–2182, 2017. 2

[23] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer, 2020. 2, 8

[24] Antoine Plumerault, Hervé Le Borgne, and Céline Hudelot. Controlling generative models with continuous factors of variations. *arXiv preprint arXiv:2001.10238*, 2020. 2

[25] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018. 2

[26] Soham Saha, Girish Varma, and CV Jawahar. Improved visual relocalization by discovering anchor points. *arXiv preprint arXiv:1811.04370*, 2018. 1, 3

[27] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixe. Understanding the limitations of cnn-based absolute camera pose regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3302–3312, 2019. 3

[28] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018. 2

[29] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013. 6

[30] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3d-structure-aware neural scene representations. In *Advances in Neural Information Processing Systems*, pages 1121–1132, 2019. 1, 2, 5

[31] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Multi-view 3d models from single images with a convolutional network. In *European Conference on Computer Vision*, pages 322–337. Springer, 2016. 2

[32] Michael Taylor. Lectures on lie groups. *Lecture Notes, available at http://www. unc. edu/math/Faculty/met/lieg. html*, 2002. 4

[33] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2595–2603, 2019. 2

[34] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. Demon: Depth and motion network for learning monocular stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5038–5047, 2017. 2

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017. 2

[36] Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, and Jakob Grue Simonsen. On position embeddings in bert. In *International Conference on Learning Representations*, 2021. 2

[37] Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Interpretable transformations with encoder-decoder networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5726–5735, 2017. 2, 3

[38] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018. 6

[39] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019. 1, 2