Evaluating Rotation Parameterizations for Orientation Estimation from RGB Images

Athrva Pandhare
University of Pennsylvania
Philadelphia, PA, USA
athrva@seas.upenn.edu

Abstract

Rotations are difficult to work with in a deep learning setting. This is primarily because rotations are not members of vector spaces and hence defining conventional derivatives is a difficult task. Rigorous calculation of the derivatives requires Lie theoretic considerations that require extra effort to implement in Deep learning. Furthermore commonly used deep learning libraries like PyTorch and Tensorflow do not readily provide instruments to calculate derivatives on Lie manifolds. In this report, we will evaluate alternate parameterizations of rotations that are either defined on tangent spaces isomorphic to vector spaces or are defined on vector spaces. We will then compare the parameterizations by evaluating performance on the downstream task of estimating relative object orientation from RGB images.

1. Motivation

Estimation of Orientation is an important task in robotic applications like state estimation, perception, manipulation [2]. More importantly, as learning based techniques for these tasks gain popularity, the task of estimation rotations from neural networks becomes important. It is important to appreciate that estimating rotations is a difficult task [4]. This is primarily because rotations are not members of ordinary vector spaces. This means that; for instance rotation matrices cannot be treated as having the same degrees of freedom as the number of elements in them. In fact, the constraints on rotations effect the degrees of freedom. It is difficult to impose these constraints onto a neural network [5], hence the most commonly used techniques assume that rotations are members of vector spaces and find the closest "valid" rotation to the predicted rotation during inference. Such techniques are not particularly efficient as can be noted from the extensive literature of Lie theoretic applications in state estimation[]. For a purely deep learning standpoint, although the operations involved in finding the closest "valid" rotation to the predicted rotation are differentiable in nature, the mapping from $\mathbb{R}^n \rightarrow \mathbb{S}^3$ or $\mathbb{R}^n \rightarrow \mathbb{SO}(3)$ is many to one. This makes the process of learning using gradient descent more problematic [4].

2. Introduction

In general, the estimation of object pose from 3D images is an important task, however, it is also a difficult task to accomplish reliably. There are multiple reasons for this, particularly, if the objects are present in cluttered environments, it becomes important to first localize the objects and infer their poses [3, 11]. In this report, we are primarily interested in the estimation of object rotation from RGB images. This choice is because estimating rotations is a more challenging task in recovering poses from images. Translations can be estimated with relative ease since there are no constraints on the translation vector [1].

Direct or Semi-Direct methods for Pose Estimation for Multiple views [7, 8] : Other traditional methods that use direct methods that rely on photometric information also do not work reliably on featureless/ textureless backgrounds. Furthermore, these methods typically require continuity in the image characteristics like brightness, contrast. In real world settings, it is difficult to impose this continuity. Additionally, modelling glare/ discontinuity in image acquisition is a difficult task to accomplish.

Learning Based Mathods for Pose Estimation : Among the Learning based methods, the deep feature detection techniques are common. These techniques are generally, not very efficient as they require multi-scale feature detection and refinement process, the same is also true for the pose estimates using such methods. Other techniques use Neural networks to directly estimate object poses from images in an end-to-end fashion [4, 9]. This is generally accomplished by adapting common object detection/ segmentation neural networks to also output a pose estimate as the output. This can be done by setting an initial identity

pose and calculating the object pose with respect to the indentity pose [5, 11]. However, the more reliable option is to use at least two images and use the relative pose transformation between the two images as the output from the neural network [2].

In this report, we do not aim to beat any of the SOTA pose detection techniques, but intend to carry out a (some what) rigourous analysis of the different rotation parameterizations that can be learned using neural networks. We will then compare performance among the different parameterizations and attempt to interpret the obtained results. We provide definitive interpretations and explanations for the superiority of some of the rotation paremeterizations over the others. In our opinion, such an analysis is valuable as it allows the readers to select a rotation parameterization based on evidence and mathematical proof.

3. Lie Groups and Tangent Spaces

We provide a very brief introduction to the topics from Lie theory that are directly relevant to the development in this report. This introduction is not designed to be complete and takes some liberty in the definition of Lie theoretic objects in the favor of simplicity.

3.1. A Lie Group :

A Lie group essentially combines the notions of a *group* and a *smooth manifold* [1].

Group : We define a group $(\mathcal{F}, \odot)$ consists of a set $\mathcal{F}$ with an operator $\odot$ [1, 10]. ($\odot$ here is not the Hadamard product but is a generic composition operator associated with the set $\mathcal{F}$). We say that if $\mathbb{A}, \mathbb{B}, \mathbb{C} \in \mathcal{F}$, and if $(\mathcal{F}, \odot)$ is a valid group, the following must be true [1].

- **Existence of Identity element $\mathcal{E}$:**

$$\mathcal{E} \odot \mathbb{A} = \mathbb{A} \odot \mathcal{E} = \mathbb{A} \tag{1}$$

- **Closure property under the operator $\odot$:**

$$\mathbb{A} \odot \mathbb{B} \in \mathcal{F} \tag{2}$$

- **Existence of the Inverse $\mathbb{A}^{-1}$:**

$$\mathbb{A}^{-1} \odot \mathbb{A} = \mathbb{A} \odot \mathbb{A}^{-1} = \mathcal{E} \tag{3}$$

- **Associativity :**

$$(\mathbb{A} \odot \mathbb{B}) \odot \mathbb{C} = \mathbb{A} \odot (\mathbb{B} \odot \mathbb{C}) \tag{4}$$

Smooth Manifold : We define a *smooth manifold* as a manifold that is differentiable and has a unique tangent space at every point [1, 6, 10]. The tangent space is vector space (or isomorphic to it) which facilitates the calculation of derivatives. This also means that a smooth manifold locally resembles a vector space (or is locally isomorphic to a vector space) [1, 6, 10]. We say that the pose of our robot (particularly the rotation) essentially evolves over this manifold [3]. Finally, the tangent space at the identity element $\mathcal{E}$ is called the *Lie Algebra* [1, 6, 10].

3.2. Actions of Lie Groups on other Sets

Elements of Lie groups can be used to transform elements of other sets, an example of this is rotations. Consider that $\mathcal{F}$ is a Lie group, and say that $\mathcal{M}$ is a set, the action of the elements of the Lie group $\mathcal{F}$ on the elements of the set $\mathcal{M}$ is [1, 6, 10],

$$\cdot : \mathcal{F} \times \mathcal{M} \to \mathcal{M} : (\mathcal{X}, v) \mapsto \mathcal{X} \cdot v \tag{5}$$

where we say that $\mathcal{X} \in \mathcal{F}$ and $v \in \mathcal{M}$. Furthermore, for $\cdot$ to be a valid action, we must have,

$$\mathcal{E} \cdot v = v \tag{6}$$

$$(\mathcal{X} \odot \mathcal{Y}) \cdot v = \mathcal{X} \cdot (\mathcal{Y} \cdot v) \tag{7}$$

For instance, consider $SO(n)$, the Lie group for Rotation matrices, the action induced by this Lie group is simply the rotation of a vector with respect to the origin in Euclidean space (i.e., $\mathbf{R} \cdot \mathbf{x} = \mathbf{R}\mathbf{x}$) [1].

3.3. Tangent Spaces of Lie Groups

Consider that we can move on the manifold of a Lie group, say we parameterize the motion with a variable t. We say that $\mathcal{K}(t)$ defines the motion of a point on the manifold $\mathcal{M}$. We define the velocity of this point as it moves on the manifold $\mathcal{M}$ as $\frac{\partial \mathcal{K}(t)}{\partial t}$. Intuitively, the velocity on any curved surface of a point residing on it is simply tangential to the surface. This is also true in the case of the point $\mathcal{K}(t)$. Any point $\widehat{\frac{\partial \mathcal{K}(t)}{\partial t}} k$, $k \in \mathbb{R}$ is a point on the tangent space of the manifold $\mathcal{M}$ at $\mathcal{K}$ [1, 6, 10]. This tangent space is denoted as $T_{\mathcal{K}}\mathcal{M}$. We further also define a special tangent space, $T_{\mathcal{E}}\mathcal{M}$, which is the tangent space at the Identity element of the Lie group. This tangent space is called the *Lie Algebra* of the group denoted as $\mathfrak{m}$ [1, 2, 6, 10, 11].

It is important to note that the *Lie Algebra*, along with other tangent spaces are all either vector spaces or are isomorphic to vector spaces [1, 6]. This means that the elements of the Lie algebra $\mathfrak{m}$ can be identified with elements of a vector space. This fact is of special interest to us, since it gives us a way of parameterizing complex objects, living on constrained manifolds with simpler elements living on vector spaces. In fact, the inter-conversion of elements living on Lie manifold to their corresponding elements on the Lie algebras of the manifold are accomplished by the following mappings [10].

3.4. The Exponential Map

The exponential map provides a way to map an element on the Lie algebra to it's corresponding element on the manifold $\mathcal{M}$ belonging to corresponding Lie group. For Lie groups, the exponential mapping is exact, and is defined as,

$$\exp : \mathfrak{m} \rightarrow \mathcal{M} \tag{8}$$

3.5. The Log Map

We define the inverse operation, that is, the mapping of points on the Lie manifold $\mathcal{M}$ to the Lie algebra $\mathfrak{m}$ as the log map.

$$\log : \mathcal{M} \rightarrow \mathfrak{m} \tag{9}$$

3.6. Derivatives of elements of Lie groups

For our analysis, it is important to recognize that the derivatives of elements the belong to the manifold $\mathcal{M}$ of a Lie groups are not trivial [4, 5, 11]. That is, consideration has to be granted to the calculation of derivatives on Lie groups. From a practical standpoint, it simply means that we cannot directly use a neural network to predict elements of a Lie group. Primarily because of the constraints on the elements of a Lie group (recall that they live on a manifold). More concretely, say we wish to predict rotation matrices $R \in SO(3)$ using a neural network. This cannot be done directly, since the space of R is not a vector space [1, 9, 10]. However, we saw that every element of a Lie manifold can be identified with an element of it's corresponding Lie algebra. Therefore, every rotation matrix R can be identified with an element of the Lie algebra of $SO(3)$, which we denote as $\mathfrak{so}(3)$. Note that we also said that a Lie Algebra is a vector space, the implication of this is that traditional derivatives are defined for elements of a Lie algebra. It is therefore possible to predict not $R \in SO(3)$, but $r \in \mathfrak{so}(3)$ and then use the exponential mapping to obtain the corresponding R ($exp(r) \rightarrow R$ where $r \in \mathfrak{so}(3), R \in SO(3)$).

This is the theme of this report, we parameterize exotic objects living on manifolds with the elements on their Lie algebras and assume that our neural network optimizes on this tangent space. This is reasonable, since tangent spaces are isomorphic to vector spaces. In the subsequent sections, we will define concrete parameterizations for denoting rotations using this technique.

4. Rotation Parameterizations

4.1. $\tilde{R}$: A Generic 3×3 Matrix

In this parameterization, we simply predict a 3×3 matrix $\tilde{\mathbf{R}}$ from the neural network. This matrix is not necessarily a valid rotation, and we assume that the model can learn to restrict the output domain to $SO(3)$ implicitly. This parameterization does not use any of the developed analysis, and will serve as the baseline for comparison.

4.2. $\widehat{R}$: Closest SO(3) neighbor to a Generic 3×3 Matrix

We first start with the naive approach. We will consider that the neural network $h(x; \theta)$ gives us $\hat{R} \in \mathbb{R}^{3\times3}$ as output. Note that this is not a Special Orthogonal matrix, and hence is not a valid rotation. To get a valid rotation R, we find the closest (in an L_2 sense) $SO(3)$ matrix to $\hat{R}$. The procedure to do this is as follows,

$$U\Sigma V^T = \hat{R} \tag{10}$$

then,

$$R = \begin{cases} UV^T & \text{if } \Delta|UV^T| = 1 \\ -UV^T & \text{if } \Delta|UV^T| = -1 \end{cases} \tag{11}$$

where $\Delta|.|$ is the determinant. The above procedure ensures that R is a valid rotation. To summarise, we use a neural network to obtain $\hat{R}$ and then get the closest special orthogonal matrix R to $\hat{R}$ using SVD.

4.3. $\omega^\times$: A member of $\mathfrak{so}(3)$

Consider the group of rotation matrices $SO(3)$, we define it's Lie algebra to be $\mathfrak{so}(3)$. $SO3$ essentially consists of 3×3 rotation matrices R [1]. We said that Lie groups have an $inverse$ operation which for rotation matrices R is the orthogonality condition,

$$R^T R = \mathbb{I} \tag{12}$$

differentiating this with respect to time gives us,

$$\frac{\partial R^T R}{\partial t} = R^T \dot{R} + \dot{R}^T R \tag{13}$$

we can notice that,

$$R^T \dot{R} = -(R^T \dot{R})^T \tag{14}$$

which means that $R^T \dot{R}$ is a skew symmetric matrix [10]. We will represent a skew-symmetric matrix with a $\times$ superscript. Therefore, consider that,

$$\omega^\times = R^T \dot{R} \tag{15}$$

Note that $\omega^\times \in \mathfrak{so}(3)$. Assuming that the initial rotation $R = \mathbb{I}$, we can define the change in rotation over time as $\omega^\times$. Note that this is also the representation of the angular velocity. The matrix $\omega^\times$ has the following form,

$$\omega^\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{16}$$

on further inspection, one can note that,

$$\omega^\times = \omega_x \mathbf{e_1} + \omega_y \mathbf{e_2} + \omega_z \mathbf{e_3} \tag{17}$$

where $\mathbf{e_1} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$, $\mathbf{e_2} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}$ and, $\mathbf{e_3} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$. This means that a rotation matrix R can be denoted be a vector $\omega = (\omega_x, \omega_y, \omega_z) \in \mathbb{R}^3$ by applying an exponential map $\texttt{exp}$ to the skew-symmetric representation of this vector [1,6,10]. This is significant because, we can use our neural network to predict this vector ω and manually apply the $\texttt{exp}$ map to get a valid rotation. This parameterization, contrary to the previous parameterization using SVD, does not have a notion of approximating R as it gives us an accurate mapping of R on the manifold by the application of the $\texttt{exp}$ map.

To recover the rotation matrix R from the vector ω, we can apply the $\texttt{exp}$ map as follows (assuming analysis for unit time interval t),

$$R = \texttt{exp}(\omega^{\times} t) \tag{18}$$

say that $\tilde{\theta} = \omega t$, meaning,

$$R = exp(\tilde{\theta}^{\times}) \tag{19}$$

this is a matrix exponential and is equivalent to,

$$R = \mathbb{I} + \mathbf{a}^{\times} sin(\theta) + \mathbf{a}^{\times}(1 - cos(\theta)) \tag{20}$$

where, $\tilde{\theta} = \mathbf{a}\theta$. Notice that this is similar to the angle axis representation with $\mathbf{a}$ being the axis and θ being the angle [1].

4.4. $\mathbf{u}v$: An element of $\mathfrak{s}^3$

Consider the group S^3, which is the group of unit quaternions [1,3,6,10]. We define the Lie Algebra for this group as $\mathfrak{s}^3$. Consider the unit norm constraint on quaternions,

$$\mathbf{q}^* \dot{\mathbf{q}} = -(\mathbf{q}^* \dot{\mathbf{q}})^* \tag{21}$$

The above form is that of pure quaternions with the real part being 0. This is important to appreciate, since this implies that the elements of the Lie algebra take a very simple form given by,

$$\mathbf{u}v = (iu_x + ju_y + ku_z)v \tag{22}$$

where $\mathbf{u}$ is pure and unitary and v is a scalar. This is simply equivalent to the following,

$$\alpha = \mathbf{u}v \tag{23}$$

where α can be interpreted as some generic vector. The consequence of this simple form of α is that it can be directly predicted from the neural network. We can easily calculate $\mathbf{u}$ and v from the predicted α as follows,

$$\mathbf{u} = \frac{\alpha}{|\alpha|} \tag{24}$$

and,

$$v = |\alpha| \tag{25}$$

The quaternion corresponding to the predicted rotation can then be obtained by applying the $\texttt{exp}$ map on α,

$$\mathbf{q} = \texttt{exp}(\alpha) = \texttt{exp}(\mathbf{u}\theta) \tag{26}$$

which evaluates to,

$$\mathbf{q} = cos(\theta/2) + \mathbf{u} sin(\theta/2) \tag{27}$$

This quaternion can then be converted to an equivalent rotation matrix as required. Note that the factor of $1/2$ is not required in the above equation since, the value of θ originates from v which is predicted from the neural network.

5. Design of the Neural Network

Having defined the different rotation parameterizations, we will define the neural network in this section. We will favour the use of simpler neural architecture to keep the focus of this report on the evaluation of the proposed rotation parameterizations. Nonetheless there are a few considerations in the design of the neural network.

- An important fact to consider is that the task we seek to accomplish is the prediction of the pose (primarily the orientation) from RGB images. This is a composite task in that the neural network must first segment the object of interest before predicting it's pose. It is of benefit to us to also predict the semantic mask of the object during pose estimation. This not only gives us the semantic location of the object at inference time, but also ensures that the neural network "focusses" on the object of interest while making pose predictions. This is especially important in cluttered environments.

- There is confusion with regards to the reference frame of the pose to be estimated. This simply means that rotations are defined with respect to some frame, and the choice of the inertial frame determines the rotation. Ideally, to counter this ambiguity, we should input two images to the neural network, one image signifies the reference and the second image signifies the query. The pose of the query is estimated with respect to the reference. In our implementation, we input the photometric difference between the query image and the reference image to the neural network, and estimate the relative pose between the query image and the reference image.

  $$\mathcal{I}^{\delta} = \mathcal{I}_r - \mathcal{I}_q \tag{28}$$

  where $\mathcal{I}^{\delta}$ is the photometric difference between the two images, $\mathcal{I}_r$ is the reference image and $\mathcal{I}_q$ is the

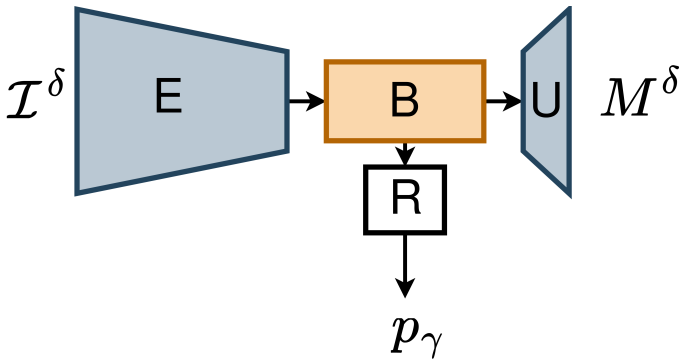

Figure 1. Simplified Model Architecture. **E** : Encoder block, **B** : Bottleneck, **R** : Rotation Head, **U** : Upsampling Block. $\mathcal{I}^{\delta}$ denotes the Photometric difference between images, M^{δ} is the difference between binary masks and p_{γ} is the relative pose prediction

query image. Furthermore, in addition to relative pose estimation, we define the auxiliary task of predicting the difference of the masks of the reference and the query image (denoted by M^{δ}). Doing so induces a kind of "soft forced" attention that makes the model focus on the difference between the reference and the query images making the pose estimation process more robust.

$$M^{\delta} = M_r - M_q \tag{29}$$

where M^{δ} is the difference between the two masks, M_r is the reference mask and M_q is the query mask.

5.1. Encoder Design

We select a simple encoder consisting of `Conv2d` and `MaxPool` layers. The role of the encoder is to reduce the image/ feature-map dimensions in successive levels while increasing the channel depth. The features at the lowest level (with the smallest spatial size and the largest channel dimensions) are then feed into the rotation head R, and the upsampling block U.

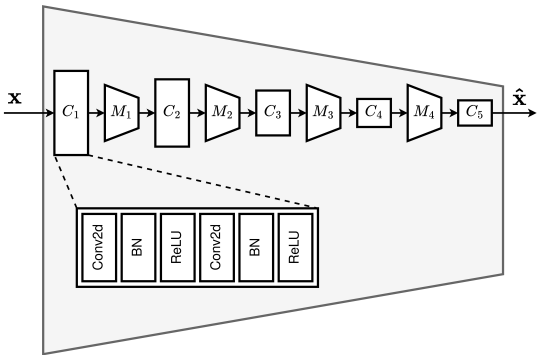

Figure 2. Encoder Design, the encoder is a simple architecture that downsamples the input images via successive application of `Conv2d` and `MaxPool` layers

The encoder design can be improved by incorporating skip connections such as in a Feature Pyramid Network. However since the purpose of this report is primarily the comparison of rotation parameterizations, we keep the network design as simple as possible.

5.2. Decoder and Bottleneck Design

The decoder is designed to be short and to contain minimum number of parameters. This is because the premise for producing the difference mask M^{δ} from the decoder is that we would like our neural network to "attend" to the areas of the image $\mathcal{I}^{\delta}$ where the photometric difference is high. Doing so will make the pose estimate more robust. This is because this allows us to ensure that the network looks at relevant parts of the image while making pose predictions. However, since the pose estimation is done from the rotation head R which receives feature maps from the bottleneck, we would like the features in the bottleneck to be as similar to the features in the decoder as possible. This ensures that the segmentation is not learned in the decoder and is learned in either the encoder or the bottleneck (thereby making sure that pose estimation is done based on features containing rich segmentation information).

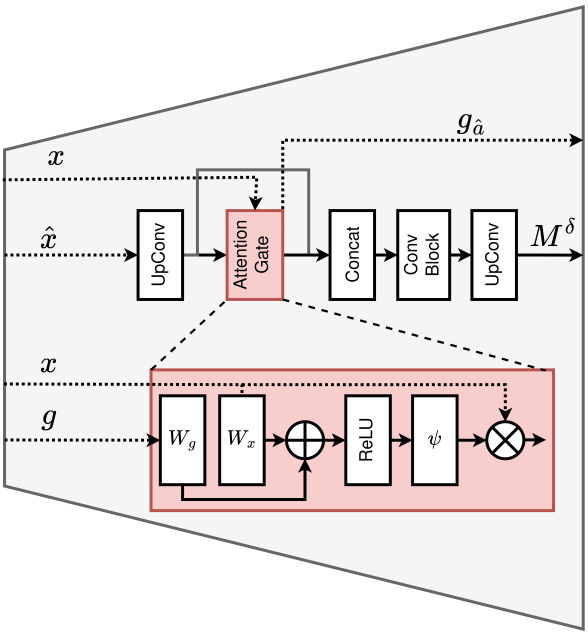

Figure 3. Decoder+Bottleneck Design, the decoder consists of `Conv2d` layers with `UpSample` layers. Notably, it contains the Attention gate from Attention UNet. The output of the Attention gate $g_{\hat{a}}$ is feed to the Rotation head, while the output of the decoder is the difference mask M^{δ}. $\hat{x}$ is the output of the encoder, x is the output of the second last layer of the encoder and, g is the ouput of the first `UpConv` layer in the decoder.

5.3. Design of the Rotation Head

Depending on the Parameterization, the output dimensionality of the Rotation head is changed. Essentially, the rotation head is a very small block that simply adapts the output of the Attention gate $g_{\hat{a}}$ in the bottleneck to the correct dimensions required by a specific rotation parameterization.

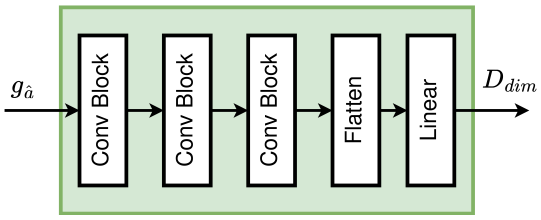

Figure 4. Rotation head design, the rotation head is a simple block that adapts features from the Attention gate output $g_{\hat{a}}$ to the dimensionality D_{dim} required by the specific rotation parameterization.

6. Loss Functions

Every rotation parameterization requires a slightly different formulation of the Loss. This is because the output (say q) that is received from the rotation head in each case is different. In some cases $q \in \mathbb{R}^n$, while in other cases these live on tangent spaces ($q \in \mathfrak{so}(3)$,$q \in \mathfrak{s}^3$). While tangent spaces are linear, to obtain the appropriate rotation matrix, different `exp` maps have to applied to these quantities. Therefore, while the essence of the loss function is the same, the formulation varies.

6.1. Loss function for $\tilde{R}$: A Generic 3×3 Matrix

We note that $\tilde{\mathbf{R}}$ is not a valid rotation. While we would like the neural network to learn to predict a valid rotation, we cannot rely on the special properties of rotation matrices to formulate a loss in this case. We therefore formulate the simplest possible loss, which seeks to minimize the L_2 norm between the predicted rotation matrix $\tilde{R}$ and the ground truth rotation matrix R^*.

$$l(\tilde{R}, R^*) = ||\tilde{R} - R^*||_2 \tag{30}$$

6.2. Loss function for $\hat{R}$: Closest SO(3) neighbor to a Generic 3×3 Matrix

In this case, the matrix $\hat{\mathbf{R}}$ is the closest $SO(3)$ neighbor to the predicted rotation matrix and is hence a valid rotation. This means that we can make use of the properties of rotations to formulate a loss function in this case. Specifically, we know that rotation matrices are special orthogonal implying that $RR^T = \mathbb{I}$. If say that R^* is the true rotation and $\hat{\mathbf{R}}$ is the predicted rotation, both of which are SO(3). Then,

$$l(\hat{\mathbf{R}}, R^*) = ||\hat{\mathbf{R}}(R^*)^T - \mathbb{I}||_2 \tag{31}$$

6.3. Loss function for $\omega^{\times}$: A member of $\mathfrak{so}(3)$

We already showed how the vector ω may be converted to a valid rotation using the `exp` map. Once the `exp` map is applied, and the predicted rotation $\mathbf{R_p}$ is found, we use the same loss as in the above case.

$$l(\mathbf{R_p}, R^*) = ||\mathbf{R_p}(R^*)^T - \mathbb{I}||_2 \tag{32}$$

6.4. Loss function for $\mathbf{u}\theta$: Direct Angle Axis representation

The conversion from the angle-axis representation to an equivalent rotation matrix is pretty standard and is differentiable. Hence the obtained angle-axis representation is first converted to a rotation matrix $\mathbf{R_{u\theta}}$ and the loss is calculated as follows,

$$l(\mathbf{R_{u\theta}}, R^*) = ||\mathbf{R_{u\theta}}(R^*)^T - \mathbb{I}||_2 \tag{33}$$

6.5. Loss function for $\mathbf{u}v$: An element of $\mathfrak{s}^3$

In this case, we saw that a quaternion is obtained. There are two ways of formulating the loss function. We can either convert the quaternion to a rotation matrix and calculate the loss as in the previous case, or we can exploit the properties

of quaternions to formulate the loss. Specifically, we know that for unit quaternions,

$$\mathbf{q}^*\mathbf{q} = 1 \tag{34}$$

where $\mathbf{q}^*$ is the conjugate. Therefore if say $\mathbf{q_p}$ is the predicted quaternion, and $\mathbf{q_{opt}}$ is the true quaternion, we can formulate the loss as follows,

$$l(\mathbf{q_p}, \mathbf{q_{opt}}) = ||\mathbf{q_p}^*\mathbf{q_{opt}} - 1||_2 \tag{35}$$

6.6. Total Loss

Assume that the above calculated formulations constitute the rotation loss, we additionally have the Mask loss that encourages the model to pay more attention to regions of high photometric difference. Say that the rotation loss is $\mathbf{L_{rot}}$ and the mask loss is $\mathbf{L}_{M^\delta}$. The mask loss $\mathbf{L}_{M^\delta}$ is the BCE Loss between the true mask difference M^δ and the predicted mask difference $\hat{M}^\delta$. The total loss is then,

$$\mathcal{L} = \mathbf{L_{rot}} + \beta\mathbf{L}_{M^\delta} \tag{36}$$

We set $\beta = \frac{1}{30}$ which is a small value. By predicting the mask difference, we want the model to pay attention to the photometric difference. This can be seen as a kind of indirect regularization of the model using the mask difference M^δ. Therefore, the magnitude of this regularization should typically be small compared to the actual objective, which is the prediction of the rotation.

7. Experimental Results

In this section we show the qualitative and quantitative results of using different parameterizations for estimating the rotation.

7.1. Datasets

We train the model on the google scanned objects dataset. These dataset is so chosen because the google scanned objects dataset has RGB images and corresponding pose information **in the absence of clutter**. Our purpose in this project is to develop formulations and compare performance between different rotation parameterizations. We would thus like to keep things simple, this is also the reason why we chose a simple architecture. Choosing a dataset that has background clutter will require a much larger model.

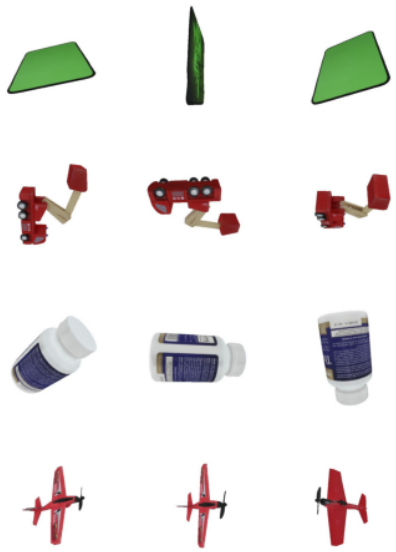

Figure 5. Sample images from the google scanned objects dataset. Each row is a different object, each column is a different orientation of the object corresponding to the row.

7.2. Qualitative Results on the Google Scanned Objects dataset

In this section, we show the qualitative results on the Google scanned objects dataset. As discussed, the model outputs are the Mask difference M^δ and the pose prediction p_γ.

7.2.1 Comparison of the Mask difference M^δ predictions

It can be seen from Fig. 6 that the soft mask difference prediction is similar for all the models. This is to be expected since we designed the model such that most of the mask information was learned in the encoder. Another thing to note here is that the design of the neural network constraints that the mask show the areas/ features that are used most form predicting the pose. The fact that the mask predictions are of good quality indicates that the regions in the images that have high photometric differences are in fact more important in the prediction of relative object poses. **We would like to re-emphasize that the mask prediction auxiliary task is like a soft regularization that nudges the model to look in (what we think) is the right place. This can be considered as a kind of direct and heavily supervised attention mechanism. The figures in Fig. 6 can be seen as soft attention heatmaps, the fact that these are so pronounced, indicates that we were perhaps able to induce supervised attention.**

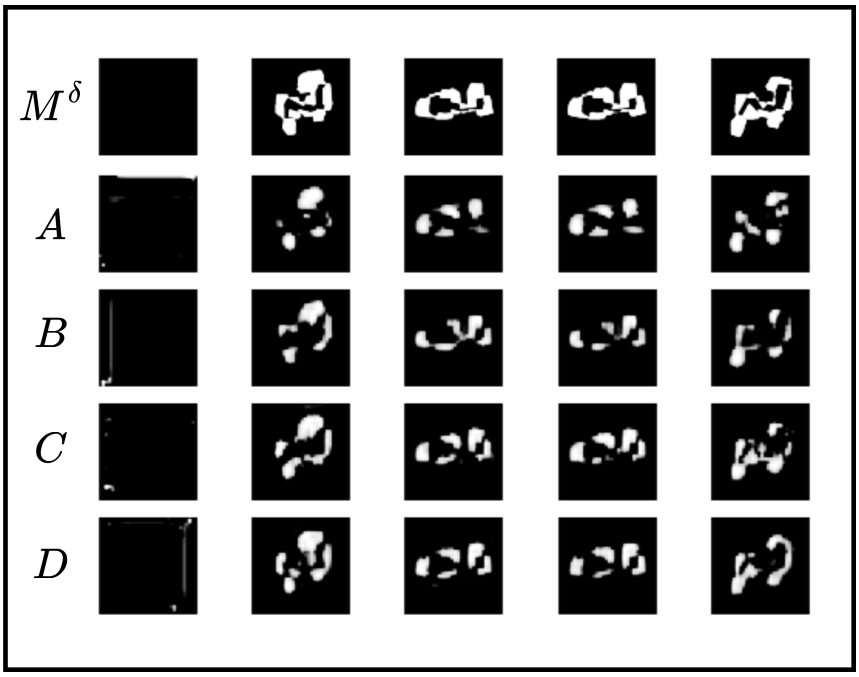

Figure 6. Top row M^δ is the ground truth mask difference, A : Mask difference using unconstrained generic matrix formulation, B : Mask difference predictions using Closest SO(3) formulation. C : mask difference prediction using $\mathfrak{so}(3)$ tangent space formulation, D : Mask difference prediction using $\mathbf{u}v$ formulation (tangent space of $\mathfrak{s}^3$)

7.3. Qualitative Comparison of Pose Estimation Performance p_γ predictions

Consider the figure Fig. 7, The orange cuboid corresponds to the predicted pose and the navy blue cuboid corresponds to the ground truth. It can be seen that the image in the first column is the reference image and all the other poses are calculated relative to the pose of this image. A corresponds to the results obtained from the formulation of $\omega^\times$ which is a member of $\mathfrak{so}(3)$, we will see quantitatively that this formulation produces the best results. B corresponds to the results obtained from the formulation of $\mathbf{u}v$ which is a member of $\mathfrak{s}^3$, from the quantitative results, we will see that this performs second best. Next, C corresponds to the 3×3 generic matrix formulation, it con already bee seen that the results obtained using this formulation are not very accurate. Finally, D corresponds to the results obtained by projecting the predicted matrix to $S0(3)$, i.e., the closest neighbor in $SO(3)$.

The qualitative results are as expected, We saw that formulations defined on tangent spaces (both $\mathfrak{so}(3)$ and $\mathfrak{s}^3$) outperform the other formulations. The formulation that considers the closest neighbor in $SO(3)$ is also very competitive, something that is somewhat unexpected due to the many-to-one mapping from to R^n to $SO(3)$. Finally, as we expected, the most naive formulation of choosing a generic matrix as a rotation and expecting the model to implicitly apply $SO(3)$ constraints performs the worst (in fact, we had to project the predicted rotation using this formulation in $SO(3)$ to get a valid rotations for visualization purposes).

7.4. Quantitative Results of Pose Estimation using different Rotation Formulations

In this section we show the quantitative results of the Pose estimation using the different rotation parameterizations. Consider Tab. 1, the results of this study were somewhat expected. We see that the formulations that assume that the neural network optimization occurs on a tnagent space $\mathfrak{so}(3) or \mathfrak{s}^3$ perform the best. It is notable that simply finding the closest neighbor in $SO(3)$ of a generic matrix (in the L_2 sense) is also very effective. As expected considering an unconstrained matrix and assuming that the model will learn the manifold constraints is not sensible. In general, considering an arbitrary matrix has unintended effects, as it defines an affine transformation instead of rotation. We can see that the error in perdiction of simply taking an affine matrix is $12\times$ taking the $SO(3)$ neighbor.

8. Conclusion

From this project, we saw that it is quite difficult to predict rotations directly from neural networks. One should always use different parameterizations that allow us to model unconstrained counterparts of rotations. In this study, we considered simple parameterizations that allowed us to be a

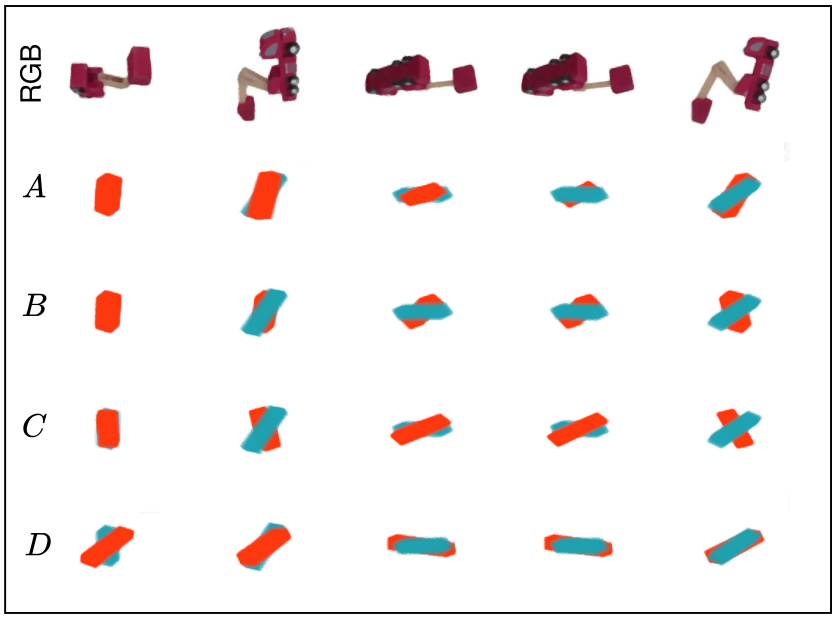

Figure 7. Top row M^{δ} is the ground truth mask difference, A : mask difference prediction using $\mathfrak{so}(3)$ tangent space formulation, B : Mask difference prediction using $\mathbf{u}v$ formulation (tangent space of $\mathfrak{s}^3$), C : Mask difference using unconstrained generic matrix formulation, D : Mask difference predictions using Closest SO(3) formulation.

Parameterizations	L_2 distance between rotations
Closest $SO(3)$ neighbor	1.415
$\omega^{\times} \in \mathfrak{so}(3)$	0.439
$\mathbf{u}v \in \mathfrak{s}^3$	0.781
Unconstrained Generic Matrix	17.857

Table 1. Performance of different Rotation Parameterizations for Pose Estimation

little clever while being in the confines of the `autograd` of Pytorch. It is possible and also a good idea to optimize on the manifold instead of using tricks (like we did). This allows a more coherent representation and is more efficient (as can be seen from a lot of Visual SLAM Literature). In conclusion, we would like to say that the formulations derived in this report are very simple, in most cases these are just linear algebra tricks. We hope that the simplicity of these formulations, along with their superiority in performance encourages others to use these in more complicated pose estimation settings.

References

[1] Hossein Abbaspour and Martin Moskowitz. Basic lie theory. 09 2007. 1, 2, 3, 4

[2] Motilal Agrawal. A lie algebraic approach for consistent pose registration for general euclidean motion. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1891–1897, 2006. 1, 2

[3] Simon L. Altmann. Rotations, quaternions, and double groups. 1986. 1, 2, 4

[4] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *European Conference on Computer Vision*, 2014. 1, 3

[5] Iryna Gordon and David G. Lowe. What and where: 3d object recognition with accurate pose. In *Toward Category-Level Object Recognition*, 2006. 1, 2, 3

[6] Roger E. Howe. Very basic lie theory. *American Mathematical Monthly*, 90:600–623, 1983. 2, 4

[7] Alexander Krull, Eric Brachmann, Frank Michel, Michael Ying Yang, Stefan Gumhold, and Carsten Rother. Learning analysis-by-synthesis for 6d pose estimation in RGB-D images. *CoRR*, abs/1508.04546, 2015. 1

[8] Frank Michel, Alexander Kirillov, Eric Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, and Carsten Rother. Global hypothesis generation for 6d object pose estimation. *CoRR*, abs/1612.02287, 2016. 1

[9] Mahdi Rad and Vincent Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *CoRR*, abs/1703.10896, 2017. 1, 3

[10] Joan Solà. Quaternion kinematics for the error-state kalman filter. *CoRR*, abs/1711.02508, 2017. 2, 3, 4

[11] Bugra Tekin, Sudipta N. Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. *CoRR*, abs/1711.08848, 2017. 1, 2, 3