Deep End-to-End Calibration Thesis (DETECT)

Ende-zu-Ende-Kalibrierung in der Robotik
Master-Thesis von Muhammad Ijlal Baig aus Rawalpindi
November 2019

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Deep End-to-End Calibration Thesis (DETECT)
Ende-zu-Ende-Kalibrierung in der Robotik

Vorgelegte Master-Thesis von Muhammad Ijlal Baig aus Rawalpindi

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Prof. Dr. Heinz Koeppl
3. Gutachten: MSc. Oleg Arenz

Tag der Einreichung:

For Fatima and Babar Mazhar Baig

Erklärung zur Master-Thesis

Erklärung zur Abschlussarbeit gemäß § 23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Muhammad Ijlal Baig, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Fall eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

In the submitted thesis the written copies and the electronic version are identical in content.

Datum / Date:	Unterschrift / Signature:
______________________	______________________________

Abstract

Hand-eye calibration is an essential operation for intelligent robots to effectively utilize their sensor data, however, traditional calibration techniques are laborious and time consuming, as they require careful setup of calibration markers [1] in the environment. The repetitive nature of the task and human administration also introduces the possibility of errors in the calibration process, which may be eliminated if the calibration and environment is memorized by a system. This thesis investigates a deep learning based approach to memorize an environment using RGBD image and uncalibrated link pose data, and perform hand-eye calibration by comparing object and pose transformations in the learnt scene. A predicted pose is considered calibrated camera pose if it is aligned with the camera axis, in which case, any transformation in the pose should also transform the image objects by a calculable amount; by comparing the consistency of the two entities we propose to find true camera pose in world frame. In this work, we use geometric transformations to calculate transform of the objects, by first projecting pixels to a point cloud, using depth information, transforming each point, and reprojecting it back to the image plane. However, our experiments show that geometric transformation of images is not a very reliable signal to learn calibration parameters. This thesis lays the foundation for future work of more sophisticated deep learning based hand-eye calibration methods, as the proposed method requires significant improvements to be applicable in real world situations.

Acknowledgments

I would like to express my sincerest gratitude to my advisor Oleg Arenz of the Intelligent Autonomous Systems (IAS) group at TU Darmstadt; without his support and valuable feedback, this thesis would not have been possible. Furthermore, I would like to thank Prof. Dr. Jan Peters, head of IAS, and Prof. Dr. Heinz Koeppl, head of Bioinspired Communication Systems (BCS) Lab, for making this thesis possible.
I would also like to acknowledge Robert Biehl, Saad Javed, and Dr. Omair Ghori for their valuable input in the implementation of this work.
Finally, I would like to thank my family and friends for their unwavering support and encouragement throughout the years of my study. Without their support this accomplishment would not have been possible.

Contents

Figures and Tables

List of Figures

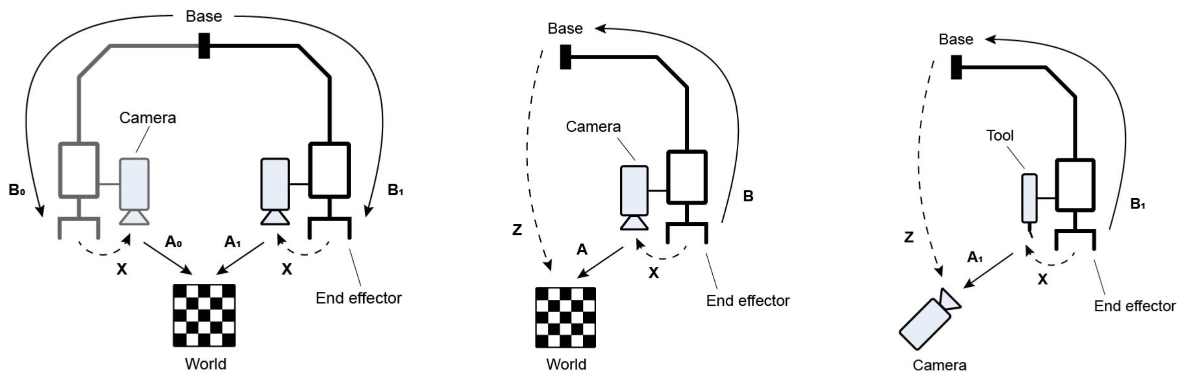
List of Tables

1 Introduction

Recent advances in robotics have attracted great contributions from industry and academia. Fueled by a dramatic increase in computational power and relatively low manufacturing costs, intelligent robots have especially become a prevalent part of both the industry and academia. Complex decision making through environment interactions is a distinctive feature of intelligent robots, which allows them to perform their tasks with speed and accuracy. These interactions are accomplished through the aid of a steady stream of data from different types of sensors; attached to the robot itself, or fixed in its work environment. However, if the robot does not have accurate knowledge of the sensor's position relative to its own frame of reference, acting on that sensor data can prove to be detrimental to the robot's operations. Therefore, it is necessary to calibrate intelligent robots with sensors that give them their environment states. The aim of a calibration is therefore, to find the transformation between the robot's frame of reference, which is usually oriented around the robot's base or its end effector, to the relevant sensor frame.
Vision sensors have especially been utilized with intelligent robots, as they offer an attractive tradeoff between precision and cost. There are also categories of vision sensors that not only provide color images, but also depth maps with decent depth resolutions. The combined depth and color data, along with recent advances in machine vision such as real-time object detection [2, 3] and segmentation [4], can help paint a more complete picture of the environment. Therefore, in our work, we target hand-eye calibration of vision sensors with depth measurement capabilities.
The most common mathematical representation of hand eye calibration consists of two basic forms, $\boldsymbol{AX} = \boldsymbol{XB}$ and $\boldsymbol{AX} = \boldsymbol{ZB}$; which are illustrated in Figure 1.1. Specifically, in Figure 1.1a, $\boldsymbol{A_i}$ represents transformation from camera frame to marker (world coordinate), $\boldsymbol{B_i}$ represents transformation from robot base frame to its end effector frame; at recording interval $\boldsymbol{i}$, and $\boldsymbol{X}$ is the unknown transformation from end effector to camera frame. It can be observed from Figure 1.1a that:

$$A_0 X B_0 = A_1 X B_1,$$
$$A_1^{-1} A_0 X = X B_1 B_0^{-1} \Longrightarrow AX = XB.$$

Multiple recordings are used to form a system of equations, which can then be solved using closed form methods and non-linear optimization. Figure 1.1b and 1.1c represent two cases of $\boldsymbol{AX} = \boldsymbol{ZB}$, where $\boldsymbol{B}$ is the transformation from end effector frame to base frame, $\boldsymbol{A}$ is the transformation from sensor frame to world coordinate, $\boldsymbol{X}$ is the unknown hand/eye transformation, and $\boldsymbol{Z}$ is the unknown robot/world transformation. This form allows simultaneous calibration of hand (end effector) to camera frame and world frame to robot base frame; which is especially useful if the camera is mounted separately from the robot, as illustrated in 1.1c.



(a) AX = XB, *hand-to-eye* configuration.

(b) AX = ZB, *eye-in-hand* configuration.

(c) AX = ZB, *hand-to-eye* configuration.

Figure 1.1.: Experimental setups for hand-eye calibration.

These traditional solutions offer accurate hand-eye calibration, if done correctly. However, setting up the calibration environment is a tedious task, and any inaccurate measurements may accumulate errors. Since link pose and camera images of the robot environment are cheap data, we are tempted explore the possibility of memorizing the room using this data. By memorizing the room environment once, and learning the calibration parameters, we could perform subsequent calibrations from memory. Thus, eliminating the need for setting up a calibration environment for future sessions in the same room. The calibration process would consist of sampling environment images from an uncalibrated camera, mounted on a robot, and querying the system for corresponding viewpoint location and orientation, henceforth mentioned as viewpoint pose, in robot's frame of reference.

A Naive approach to calibration through environment memorization would be to form a 3D point cloud context of the scene; as proposed in [5], and then correlate query images against the 3D context from various viewpoints to estimate the correct viewpoint pose. Although, this method should generate the correct viewpoint, it requires a high number of iterations to solve for each query image. This can be attributed to the fact that range of values for each *degree of freedom (DOF)* in the viewpoint pose has to be explored, to find the optimal correlation viewpoint. A simple 6D viewpoint representation consisting of 3 translation values in Cartesian coordinates and 3 Euler angles with N sample points for each DoF would require N^6 viewpoint correlations. Another issue with plain correlation is that it does not take into consideration scene semantics; a simple change in lighting conditions might affect the accuracy of the system. Ideally, we would want a system that not only memorizes the environment, but also understands semantic meaning of elements in the environment.

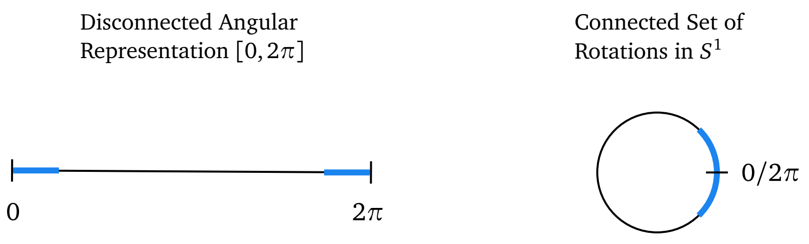
Deep learning has shown effective capability in interpreting data semantics [6, 7, 8], often with superhuman accuracy; and with optimization, these systems can also make predictions in real-time. Therefore, we were motivated to explore deep learning models as a solution to the viewpoint estimation problem. In our work, we provide link pose as latent to an encoder-decoder architecture and show that it is better at learning than traditional AE. We also explore the possibility of using geometric image transformations to learn camera calibration parameters.

2 Foundations

In this chapter we discuss a few concepts that lay foundation to the composition of our algorithm. In Section 2.1, we discuss the problems that occur while training a network on regression targets with discontinuities, and highlight a method to convert piecewise continuous rotation representation to a fully continuous form. In Section 2.2, we discuss mixup, a regularization technique that helps avoid network overfitting, and as a consequence trains more robust models. Section 2.3 highlights generative modelling, which are unsupervised techniques known for capturing underlying relations in data. In Section 2.4, we discuss projective geometry and techniques to manipulate geometry in 3D space.

2.1 Continuous Rotation Representation

Standard rotation representations like euler, quaternion and axis-angles map orientations to a piecewise continuous space, as shown in Figure 2.1. Although, it is possible to approximate a real and piecewise continuous function using neural network approximation theory, Llanas et al. [9] points out that gradient descent on a piecewise continuous function would require many more neurons and training iterations. It is worth mentioning that piecewise continuous orientation representations like quaternions [10, 11, 12] and axis angles [13, 14, 15] have been successfully regressed in previous works, but they yield higher errors at certain orientations. Xiang et al. [12] report high errors at predictions near $\pi/2$ and π, and attibute these errors to rotation ambiguity of symmetric shapes. However, Ummenhofer et al. [13] report high errors at these rotations even for non-symmetric shapes, and conclude that these errors arise due to discontinuities in the rotation representations.



Disconnected Angular Representation $[0, 2\pi]$
0
2π
Connected Set of Rotations in S^1
$0/2\pi$

Figure 2.1.: A Simple illustration highlighting the discontinuous nature of euler rotation representation in the first panel. The second panel shows continuous nature of a set of rotation, in actual space S^1.

A fully continuous rotation representation can be formed by mapping euler rotations to their *Sine* and *Cosine* pair, as proposed by Csiszar et al. [16]. The sinusoidal representation $[x, y]^T = [\cos\theta, \sin\theta]^T$, can be converted back to euler representation by taking atan2(y, x), which returns an angle θ such that $-\pi < \theta \leq \pi$.

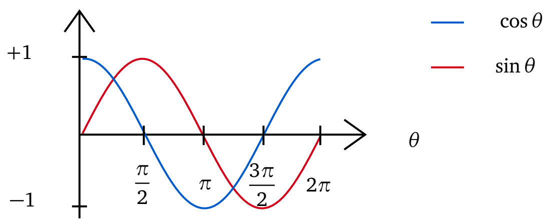


Figure 2.2.: Plot of orthogonal sinusoids, which as a pair describe unique euler angles in a continuous form. After every 2π interval, both sinusoids repeat, which removes any discontinuities found in the standard Euler representation.

2.2 Manifold Mixup

Manifold mixup [17] is a regularization technique that helps avoid overfitting in neural networks. As the availability of processing power has become more accessible, deep learning research has made great strides, utilizing networks with a

large number of parameters. Although, a larger number of network parameters increases the network's expressive power, it also makes the network more prone to overfit the training data. The overfit network has complex decision boundaries with sharp transitions that may miscategorize a datapoint with high confidence. On the contrary, a well generalized network has soft transition between decision boundaries, where probability of datapoint belonging to a class is low, furthermore, datapoints sampled uniformly in the region are distributed without bias towards any class boundary.
Manifold mixup achieves generalizability by introducing semantic interpolation as an additional training metric to produce flatter hidden representations. Consider training a deep neural network $f(x) = f_k(g_k(x))$, where g_k is a mapping from input to an intermediate hidden state at layer k, and f_k is the mapping from hidden state at layer k to the output y. *Manifold mixup* suggests selecting a layer k from a set of layers S in the network, and process two minibatches (x, y) and (x', y') until layer k, where (x', y') can be obtained by shuffling the batch rows. The hidden representations from the k^{th} layer processed inputs, and the outputs are then mixed to produce a mixed minibatch $(\tilde{g}, \tilde{y})$:

$$(\tilde{g}, \tilde{y}) = (\mathbf{Mix}_\lambda(g_k(x), g_k(x')), \mathbf{Mix}_\lambda(y, y')),$$

where $\mathbf{Mix}_\lambda(a, b) = \lambda \cdot a + (1 - \lambda) \cdot b$, and λ is the mixing coefficient obtained from a beta distribution $\lambda \sim Beta(\alpha, \alpha)$ as proposed by Zhang et al. in [18]. The forward pass is then continued from $k^t h$ layer using mixed minibatch $(\tilde{g}, \tilde{y})$, and a loss is computed using:

$$\mathcal{L}_{mixup} = \ell(f_k(\mathbf{Mix}_\lambda(g_k(x), g_k(x'))), \mathbf{Mix}_\lambda(y, y')),$$

where ℓ can be any appropriate distance measurement mertric, like mean squared error (MSE). This mixup loss is then back propagated through the entire computation graph to optimize the network.

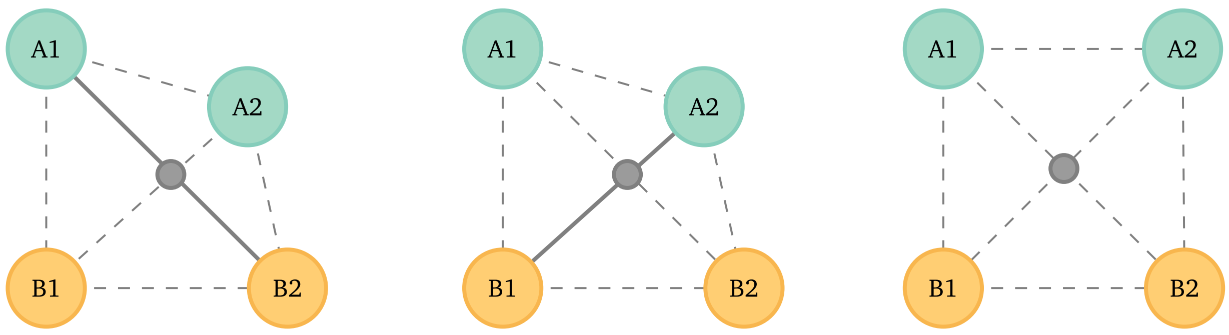


Figure 2.3.: Illustration of how manifold mixup learns flatter representation of data samples. The gray dot represents an interpolated sample at hidden layer k, formed by a mixture of hidden samples A1 and B2, shown in the left panel, which produce a label of 50% teal and 50% orange, despite being closer to the teal boundary. The same interpolated sample produced by mixture of hidden samples A2 and B1, in the middle panel, produces a label of 67% teal and 33% orange. Manifold mixup encourages consistency in label predictions of interpolated samples, regardless of the hidden samples mixed to reach interpolation point, therefore the hidden samples are rearranged in a flatter constellation, as shown in the right panel

2.3 Representation Learning

Complex observation data can be simplified by neural networks to a compressed representation, only containing the essential information required for prediction. Generative models offer an attractive approach to learn compressed representation in an unsupervised manner.

2.3.1 Generative models

Given some observations $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=0}^{N-1}$ generative systems aim to model the underlying data distribution $p(\mathbf{x})$, from which the observations were sampled. The true data distribution is often very complex; in order to simplify the modelling process, it is assumed that each data point $\mathbf{x}^{(i)}$ is sampled independent of the other, and that the underlying generative process $p(\mathbf{x})$ remains unchanged for all acquired samples. In such a case, the observations D are said to be *i.i.d.*, i.e. *independent and identically distributed*, and allows us to write the joint probability of all observations as:

$$p(\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, ..., \mathbf{x}^{(N-1)}) = \prod_{i=0}^{N-1} p(\mathbf{x}^{(i)}),$$

$$\log p(\mathcal{D}) = \sum_{x \in \mathcal{D}} \log p(\mathbf{x}).$$

Unfortunately, even with the *i.i.d.* assumption, due to high dimensionality, and interdependency of dimensions within the observation variable $\mathbf{x}$, the data distribution can still be difficult to model. Therefore, it is often desirable to model a lower dimensional distribution $p(\mathbf{z})$, which captures the essential properties of the true generative process $p(\mathbf{x})$, in a predictable and well behaved form. Samples from this latent distribution $p(\mathbf{z})$, also termed as latent code $\mathbf{z}$, should map to the observation variable $\mathbf{x}$ in a bijective manner; this ensures that the latent distribution captures meaningful properties of the true distribution.

Autoencoders

Autoencoders (AE) are an unsupervised approach to learn a generative process $p(\mathbf{x})$. AE consists of an encoder network, which maps observation samples $\mathbf{x}$ to a latent codes $\mathbf{z}$, and a decoder network that maps the latent codes $\mathbf{z}$ to an observation variable approximation $\hat{\mathbf{x}}$. A loss is then computed based on how dissimilar the approximation $\hat{\mathbf{x}}$ is from the true value $\mathbf{x}$, and backpropagated. Standard autoencoders have a bottleneck layer, which produces latent codes $\mathbf{z}$ that are set smaller in dimension compared to the observation variable $\mathbf{x}$; the bottleneck forces the network to learn only essential features. Correlated features in the observation variable $\mathbf{x}$, are mapped to the latent codes $\mathbf{z}$ with uncorrelated features. Thus, standard autoencoders work similar to Principal Component Analysis (PCA). However, neural networks can represent non-linear mappings between the observations $\mathbf{x}$ and latent variable $\mathbf{z}$, and thus, can capture more complex relations than the standard PCA.

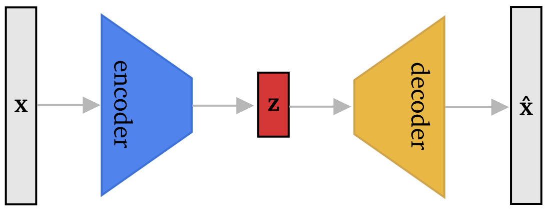


Figure 2.4.: Illustration of the standard autoencoder architecture, where the encoder maps the observation variable x to a latent variable z; the decoder constructs x̂, which is an approximation of the original observation variable x.

Variational Autoencoders

Variational Autoencoders (VAE) are an unsupervised approach to simultaneously learn a generative processes $p(\mathbf{x})$ and a corresponding latent distribution $p(\mathbf{z})$, utilizing visual inference and gradient descent. It consists of an encoder, or inference network, which estimates the true posterior distribution $p(\mathbf{z}|\mathbf{x})$, with an approximate $q(\mathbf{z}|\mathbf{x})$, and a decoder, or generative network that maps latent variable samples $\mathbf{z}$ back to observation samples $\mathbf{x}$. Backpropagation is performed through both networks, treating them as a single unit, which helps optimize both the inference and generator network in tandem, and fine tunes an approximate posterior distribution $q(\mathbf{z}|\mathbf{x})$.
As shown in Appendix B, we can derive the following equation from the log of data distribution $p(\mathbf{x})$, also termed as evidence:

$$\log p(\mathbf{x}) - \underbrace{(E[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})])}_{ELBO} = \underbrace{D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})]}_{error\ in\ approximation}, \tag{2.1}$$

Equation (2.1) can be viewed as a composition of three basic terms; the log evidence, or true data distribution, the *evidence lower bound (ELBO)*, or inferred approximation of the log evidence, and the error term resulting from the inferred approximation. The error term, defined as KL-divergence between inferred posterior $q(\mathbf{z}|\mathbf{x})$ and true posterior $p(\mathbf{z}|\mathbf{x})$, reveals the quality of our approximation and should be minimized to acquire a good evidence approximation. However, the dimensionality of our observations $\mathbf{x}$, and even the latent variable $\mathbf{z}$ in some cases can be qiute large, rendering the posterior intractable. Instead, we can maximize the *ELBO*, which brings the evidence lower bound closer to the true evidence, and as a result the approximation error is minimized indirectly. The *ELBO* consists of term $E[\log p(\mathbf{x}|\mathbf{z})]$, maximizing which, can be viewed as performing maximum likelihood estimation, and term $D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})]$, which is KL-divergence between the approximated posterior distribution $q(\mathbf{z}|\mathbf{x})$ and a prior $p(\mathbf{z})$ of our choosing; it encourages our posterior to assume the distribution of our chosen prior. In standard VAE, Gaussian distribution is used to approximate the posterior distribution $q(\mathbf{z}|\mathbf{x})$; an easy choice of prior $p(\mathbf{z})$ is the normal distribution $\mathcal{N}(0,1)$. Since Gaussian distributions are self-conjugate, choosing a Gaussian prior results in a convenient, closed-form posterior. The term used to constrain approximated posterior distribution $q(\mathbf{z}|\mathbf{x})$ to chosen prior $p(\mathbf{z})$ can be written as:

$$D_{KL}[\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1)] = \frac{1}{2}\sum_k \left(\sigma + \mu^2 - 1 - \log\sigma\right), \tag{2.2}$$

where k is the dimension of latent code $\mathbf{z}$, μ and σ are mean and variance, the parameters of approximated Gaussian posterior $q(\mathbf{z}|\mathbf{x})$. In practice, it is preferred to model $\log\sigma$ instead of σ to ensure positivity of a given output. Therefore, Equation (2.2) can be written as:

$$D_{KL}[\mathcal{N}(\mu,\sigma)||\mathcal{N}(0,1)] = \frac{1}{2}\sum_k \left(\exp\sigma + \mu^2 - 1 - \sigma\right),$$

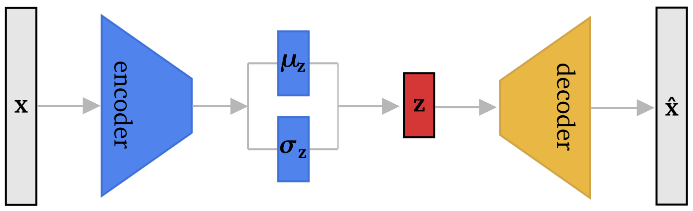


Figure 2.5.: Illustration of standard variational autoencoder architecture, where the encoder predicts parameters (μ_z, σ_z) of posterior distribution $q(\mathbf{z}|\mathbf{x})$, from which $\mathbf{z}$ is sampled. The decoder then constructs $\hat{\mathbf{x}}$, which is an approximation of the original observation variable $\mathbf{x}$.

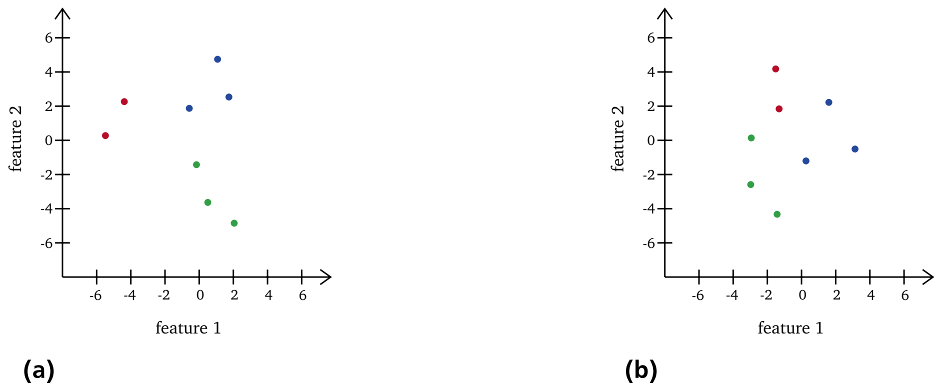


Figure 2.6.: Figure 2.6a illustrates posterior distribution predictions $q(\mathbf{z}|\mathbf{x})$ mapped to feature space, where a solid point represents mean value, while gradient region around it represents variance in the prediction. It can be observed that similar data samples are clustered together in feature space, however, the clusters may be seperated by regions which do not correspond to valid data samples. Therefore, interpolation between points in feature space landing in the invalid region would reconstruct unrealistic samples. Figure 2.6b illustrates the effect of constraining a posterior distribution $q(\mathbf{z}|\mathbf{x})$ to a Gaussian prior $\mathcal{N}(0,1)$, as in standard VAE. The data samples in feature space are grouped together within the prior distribution, therefore, reducing the invalid region between clusters.

2.4 Projective Geometry

Projective geometry is the study of the relationship between geometric objects as a result of projecting them onto other surfaces. It is extensively used in extracting and manipulating data captured by a camera image, since images are basically a projection of 3D world onto a 2D plane. The standard euclidean geometry is insufficient to study these projections, as it can be observed that parallel lines in the 3D world do not always appear parallel, rather they often intersect at some point in view; Euclidean geometry forbids intersection of two parallel lines under any circumstances, which renders it insufficient to fully explain the geometric relationships that we observe. On the contrary, projective geometry takes into account the perspective projection parameters, which cause the intersection of parallel lines.The intersection points themselves are known as vanishing points, which are located at infinity. This highlights another shortcoming of euclidean geometry, which uses cartesian coordinate system to represent positional information; the cartesian coordinate system does not have a general representation of infinity. Projective geometry on the other hand, utilises homogeneous coordinate system to conveniently represent the points at infinity.

2.4.1 Homogeneous Coordinates

Consider a point $[x, y]^T$ in 2D cartesian coordinate system. The point can be defined in homogeneous coordinates as $[wx, wy, w]^T$, where w is any non-zero number, and is referred to as the scale factor. It is worth noting that a single point in cartesian coordinates can be represented by infinite many homogeneous coordinates, since a different values of scaling factor w would change the position in homogeneous coordinates. Therefore, points and lines in 2D cartesian system become lines and planes in 3D homogeneous coordinates, respectively. Homogeneous vector $[wx, wy, w]^T$ can be converted back to cartesian coordinates by first normalizing with the scale factor w to a form $[x, y, 1]^T$, and then removing the extra coordinate. Unlike cartesian coordinate system, homogeneous coordinates have a standard method to represent points at infinity; by replacing scaling factor 1 in normalized coordinates with 0, we end up with a vector of the form $[x, y, 0]^T$, which represents infinity in a bounded form that is especially useful when performing transformation.

2.4.2 Homogeneous Transforms

Homogeneous transformations are the transformations performed on geometric elements in homogeneous coordinate system. Homogeneous coordinates have the added benefit that all transformations can also be performed using matrix multiplication, which offers a means of merging multiple transformations into a single, neatly packaged matrix. Transformations can be rigid, where the size and shape of geometry remains unaffected, like translation and rotation; or it can be non-rigid, where the shape or size of geometry is modified, like scaling and shearing. Non-rigid transformations play a significant role in projective geometry, as they define how parallel line projections converge.

2.4.3 Camera Projection

An image is the projection of geometry in 3D world onto a 2D plane, through the use of a camera. The operation can be written as:

$$\mathbf{x} = \mathbf{P}\mathbf{X},$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

where **x** is 2D image point in homogeneous coordinates, **P** is the projection matrix that defines geometric transformations taking place in the conversion from 3D to 2D, and **X** is 3D world position of point in homogeneous coordinate system. It is worth mention here that whenever a transformation is applied to a point, it must be renormalized to obtain cartesian coordinates. The projection matrix can be decomposed into two basic components, the intrinsics that define transformations taking place from camera origin to image plane, and the extrinsics that define rigid transformations, which convert a point from world frame to camera frame. A standard decomposition might look like:

$$\mathbf{P} = \underbrace{\begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{intrinsics} \underbrace{\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}}_{extrinsics}, \tag{2.3}$$

where f_x and f_y is focal length of camera along x and y-axis respectively, c_x and c_y are the pixel coordinates of image centered on the camera axis, $\mathbf{R}$ is a $3x3$ rotation matrix, $\mathbf{t}$ is a $3x1$ translation vector and $\mathbf{0}^T$ is a $1x3$ vector of zeros. It is worth mentioning here that the intrinsic parameters are dependent on camera properties, and therefore the intrinsic matrix may vary from 2.3. Appendix A lists some common transformation matrices used to compose the overall camera projection matrix, and may not always have the form in Equation 2.3.

2.4.4 Geometric Transformation

Partial 3D geometry can be reconstructed as a point cloud from a certain viewpoint if RGB-D image data from the viewpoint is available and camera parameters are known. The process can be used to approximate a viewpoint shift. In order to estimate a neighbouring viewpoint, a point cloud can be formed by projecting pixels from image plane to 3D space, a defined rigid transformation can then be applied, and the transformed point cloud can be reprojected back to the initial image plane; thus generating an estimate of a new viewpoint of known transformation from the initial RGB-D image viewpoint.

Depth Map to Point Cloud

The first step to perform geometric transformation is to project image data to 3D space, using the known camera parameters and the image depth map; the required camera parameters include (c_x, c_y), the principle points along x and y axis, maximum range of the depth sensor, and (f_x, f_y), the focal length along x and y axis, in pixels. Algorithm 1 describes the process of projecting depth map to a point cloud in 3D space.

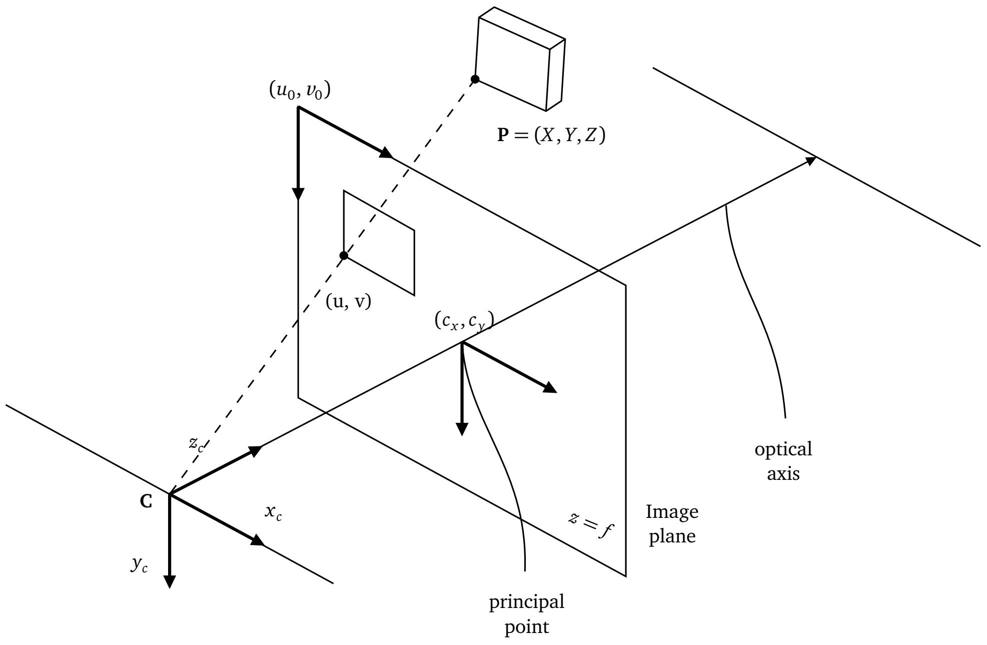


Figure 2.7.: Illustration of pinhole camera model, demonstrating projection of a point P in 3D space to the image plane at location (u, v) in image coordinates. Location of the image plane is considered at a z distance equal to focalength f from the camera origin **C**.

```
Data:
$c_x, c_y, f_x, f_y, depth\_scale$: camera parameters;
$depth$: depth map;
Result: $P$: point cloud;
for $u = 0$ to $image\ height$ do
    for $v = 0$ to $image\ width$ do
        $Z = \frac{depth(u,v)}{depth\_scale}$;
        $X = (u - c_x)\frac{Z}{f_x}$;
        $Y = (v - c_y)\frac{Z}{f_y}$;
        $P(u,v) = \{X, Y, Z\}$;
    end
end
```

Algorithm 1: Depth map to point cloud conversion

Point Cloud to Image Reprojection

In order to calculate the new transformed image, we project points from 3D point cloud to the image plane, using the same camera parameters as those used to project them out. Algorithm 2 lists the steps involved in reprojecting image back from point cloud data.

```
Data:
$c_x, c_y, f_x, f_y, depth\_scale$: camera parameters;
$P$: point cloud;
$I$: RGB image;
Result: $I'$: reprojected RGB image;
for $u = 0$ to $image\ height$ do
    for $v = 0$ to $image\ width$ do
        $X, Y, Z = P(u,v)$;
        $u' = X\frac{f_x}{Z} + c_x$;
        $v' = Y\frac{f_y}{Z} + c_y$;
        if $u' < height$ and $v' < width$ then
            $I'(u,v) = I(u',v')$
        else
            $I'(u,v) = 0$
        end
    end
end
```

Algorithm 2: Point cloud to image reprojection

3 DETECT

In our system, we have knowledge of the link pose that our camera sensor is attached to. We also assume that the camera sensor is equipped with depth measurement capability, hence, we also have RGB-D image data, corresponding to each recorded link pose. We aim to learn a model, which takes an RGB image of the environment that it learnt, and predicts the camera pose corresponding to the image view, in world coordinates. In Section 3.1, we discuss latent code representation, a vector that depicts the pose transformations in our system. Section 3.2 describes the details of our proposed loss functions, which we use to learn the robot environment, as well as, predict calibration parameters. Section 3.3 highlights the design details of our proposed algorithm.

3.1 Latent Code Representation

In DETECT, we represent the viewpoint pose as a 9D vector $\mathbf{v}$, where the vector can be decomposed to translation components $[t_x, t_y, t_z]^T$ and continuous rotational components $[s_x, c_x, s_y, c_y, s_z, c_z]^T = [\sin\theta, \cos\theta, \sin\phi, \cos\phi, \sin\psi, \cos\psi]^T$, as described in Section 2.1. Here, θ, ϕ and ψ represent rotations along axes x, y and z, respectively. Although, the rotational component can be compressed further to a 5D vector, while maintaining the continuous nature [19], the Gram-Schmidt approach used to formulate a 5D vector increases the computational complexity. Since we require frequent conversions between the continuous and Euler rotational representations during the training process, preference was given to the 6D rotational representation proposed by Csiszar et al. [16].

3.2 Loss Formulation

Our proposed calibration system consists of an encoder network E, parameterized by ϵ, which learns a mapping $X \mapsto V$, where X and V denotes image and link pose space, and a link-to-camera transformation $\mathbf{T}_l^c$ whose parameters are learned directly. Two transformation operators $\sigma(.)$ and $\pi(.)$ are utilized, which are differentiable in nature. The geometric transformation operator performs a mapping $\sigma(\mathbf{x}, \mathbf{d}, \mathbf{K}, \mathbf{p}) \mapsto \mathbf{x} \circ \mathbf{p}$, where $\mathbf{x}$ is an image, $\mathbf{d}$ is its corresponding depth map, $\mathbf{K}$ is camera intrinsic parameters, and $\mathbf{p}$ is a transformation vector, and $\mathbf{x} \circ \mathbf{p}$ indicates geometric transformation of image $\mathbf{x}$ by $\mathbf{p}$. The pose transformation operator performs a mapping $\pi(\mathbf{v}, \mathbf{p}) \mapsto \mathbf{v} \circ \mathbf{p}$, where $\mathbf{v}$ is a pose vector, $\mathbf{p}$ is a transformation vector, and $\mathbf{v} \circ \mathbf{p}$ indicates local pose transformation. Image instances $\mathbf{x}^{(i)}$ are mapped to link poses $\hat{\mathbf{v}}_l(\mathbf{x}^{(i)})$ by the encoder network, and link-to-camera transformation is applied to acquire predicted camera pose $\hat{\mathbf{v}}_c(\mathbf{x}^{(i)})$.

A viewpoint in space can be identified by a pose with respect to some world frame, or an image from that vantage point, where the image is considered projection of world onto a plane that is at a constant transformation from the camera origin. A calibrated system is one in which the viewpoint pose and camera origin are aligned. This implies that at any viewpoint, a particular transformation $\mathbf{p}$ in image geometry, which corresponds to camera pose transformation, would represent the same new viewpoint as original viewpoint pose transformed by the $\mathbf{p}$. This relationship can be formalized as:

$$\mathbf{v}(\mathbf{x} \circ \mathbf{p}) = \mathbf{v}(\mathbf{x}) \circ \mathbf{p}, \tag{3.1}$$

Where $\mathbf{v}(\mathbf{x} \circ \mathbf{p})$ represents viewpoint pose of geometrically transformed image $\mathbf{x} \circ \mathbf{p}$, and $\mathbf{v}(\mathbf{x}) \circ \mathbf{p}$ represents viewpoint pose of original image, pose transformed by $\mathbf{p}$. Figure 3.1 provides an overview of the system architecture.

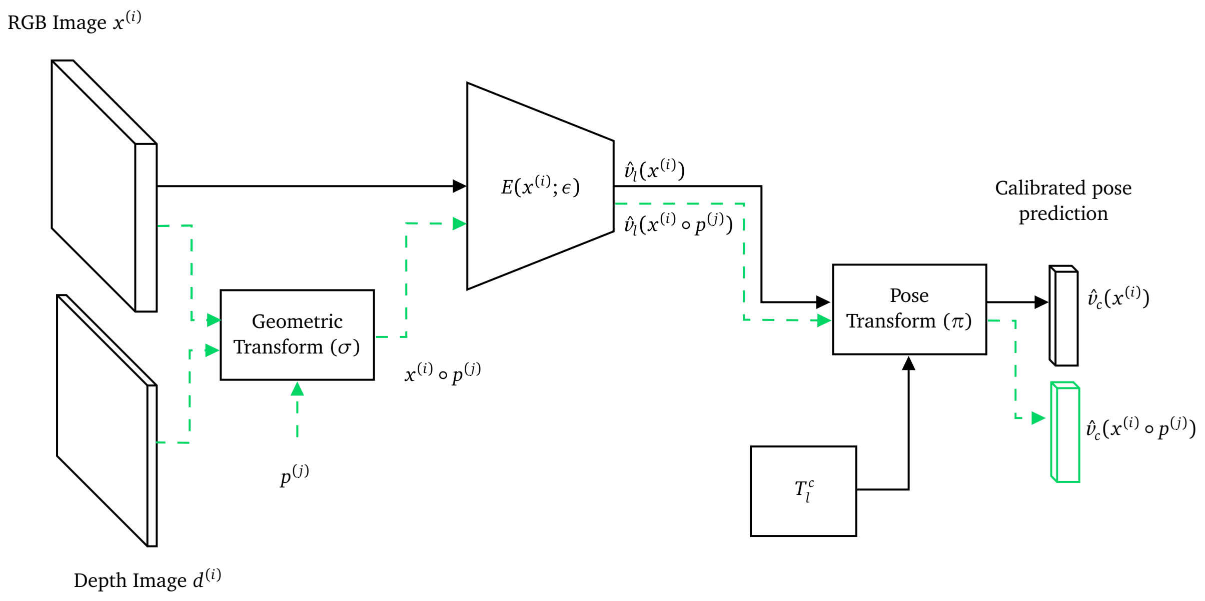


Figure 3.1.: An illustration of the architecture for the DETECT algorithm.

3.2.1 Pose Reconstruction loss

To perform calibration, our system needs to be able to probe link poses against unique images, which may not be part of the training data, therefore, it is beneficial to establish a general understanding of the environment, and the relationship between different viewpoint images and link poses. Hence, we utilize an encoder network $E(.,\epsilon)$ that learns to map image instances $\mathbf{x}^{(i)}$ to corresponding link poses $\mathbf{v}_l^{(i)}$; once optimized, the network becomes capable of predicting link pose to new viewpoint images from the environment. To find the optimal parameters ϵ, we propose a pose reconstruction loss:

$$\mathcal{L}_l(\epsilon;\mathcal{D}) = \sum_{(\mathbf{x}^{(i)},\mathbf{v}_l^{(i)})\in\mathcal{D}} \|\hat{\mathbf{v}}_l(\mathbf{x}^{(i)}) - \mathbf{v}_l^{(i)}\|_2 + \mathcal{L}_{mixup}(\xi,\zeta;\mathcal{D}),$$

where $\hat{\mathbf{v}}_l(\mathbf{x}^{(i)})$ is the link pose predicted from image instance $\mathbf{x}^{(i)}$, and $\mathcal{L}_{mixup}(\xi,\zeta;\mathcal{D})$ is the mixup loss as proposed in Section 2.2. The objective function for the encoder network can be represented as:

$$\min_{\epsilon\in F} \mathcal{L}_l(\epsilon;\mathcal{D}),$$

where F is the parameter domain for ϵ.

3.2.2 Calibration Loss

Once we have a link pose prediction, we apply the link-to-camera transformation $\mathbf{T}_l^c$ and estimate a camera pose $\hat{\mathbf{v}}_c(\mathbf{x}^{(i)})$ for image $\mathbf{x}^{(i)}$. However, if the link-to-camera transformation is inaccurate, the predicted camera pose would have a fixed error offset $\mathbf{t}$ from the true camera pose, which can be written as:

$$\hat{\mathbf{v}}_c(\mathbf{x}^{(i)}) = \mathbf{v}_c(\mathbf{x}^{(i)})\mathbf{t} \tag{3.2}$$

In order to eliminate the error offset $\mathbf{t}$, we propose the calibration loss based on Equation 3.1:

$$\mathcal{L}_c(\mathbf{T}_l^c) = \|\hat{\mathbf{v}}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}) - \hat{\mathbf{v}}_c(\mathbf{x}^{(i)}) \circ \mathbf{p}^{(j)}\|_2,$$

with an objective function:

$$\min_{\mathbf{T}_l^c \in K} \mathcal{L}_c(\mathbf{T}_l^c),$$

Since we do not know the true camera pose $\mathbf{v}_c(\mathbf{x}^{(i)})$ for an image $\mathbf{x}^{(i)}$, we cannot directly optimize link-to-camera pose transformation $\mathbf{T}_l^c$. Instead, we use the intuition from Equation 3.1, and compare the pose of geometrically transformed image against transformation of transform of pose from the original image. If the link-to-camera pose transformation $\mathbf{T}_l^c$ is inaccurate then for each camera pose prediction would have a fixed error offset of $\mathbf{t}$; therefore, we can write the camera pose estimation of an image $\mathbf{x}^{(i)}$ and its geometrically transformation projection $\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}$ as:

$$\hat{\mathbf{v}}_c(\mathbf{x}^{(i)}) = \mathbf{v}_c(\mathbf{x}^{(i)})\mathbf{t}, \tag{3.3}$$

and

$$\hat{\mathbf{v}}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}) = \mathbf{v}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)})\mathbf{t}, \tag{3.4}$$

where $\mathbf{p}^{(j)} \in P$, and P is a predefined set of perturbation transformations that ensure more than 50% of the geometry from the original image $\mathbf{x}^{(i)}$ is visible in the geometrically transformed image $\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}$. This ensures that encoder has sufficient image information to make a link pose prediction. The right-hand side of Equation 3.4 can be rewritten based on Equation 3.1 as:

$$\hat{\mathbf{v}}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}) = \mathbf{v}_c(\mathbf{x}^{(i)})\mathbf{p}^{(j)}\mathbf{t}, \tag{3.5}$$

and from Equation 3.3 we can rewrite Equation 3.5 as:

$$\hat{\mathbf{v}}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}) = \hat{\mathbf{v}}_c(\mathbf{x}^{(i)})\mathbf{t}^{-1}\mathbf{p}^{(j)}\mathbf{t}. \tag{3.6}$$

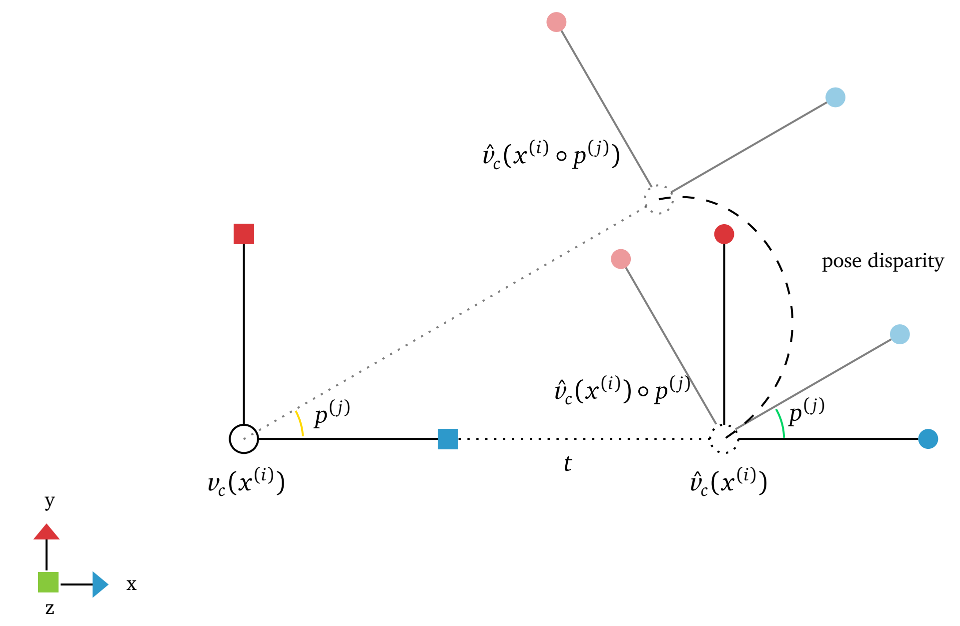
By comparing Equation 3.6 with Equation 3.1 we can say that our predicted camera pose is equal to the true camera pose if $\mathbf{t}^{-1}\mathbf{p}^{(j)}\mathbf{t} = \mathbf{p}^{(j)}$, i.e. $\mathbf{t}$ is identity transformation. Otherwise, there would be a disparity between camera pose estimation of geometrically transformed image and local transformation of image camera pose, as illustrated in Figure 3.2

3.3 Network Design

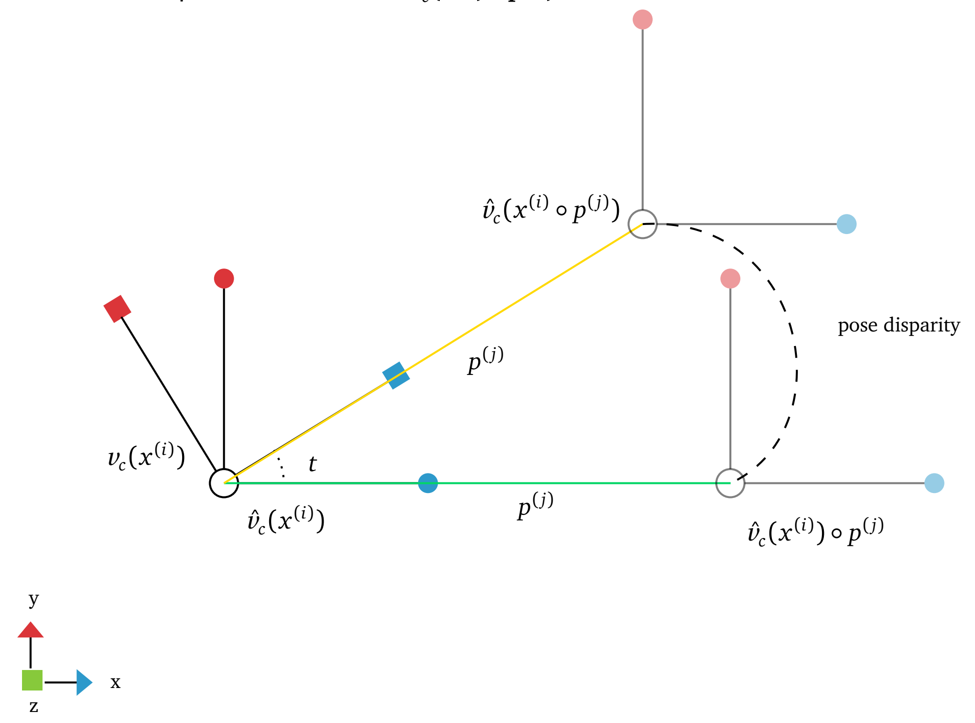
Our goal is to learn the environment surrounding a robotic arm, based on images acquired from a camera, mounted on one of the arm links, and to find the true camera pose in world frame, from image, depth and link pose data. The high level architecture is presented in Figure 3.1, with further details of the design in Appendix C and D. We formulate the following algorithm to perform end to end camera calibration.

```
Data: ${\mathbf{x}^{(i)}, \mathbf{d}^{(i)}, \mathbf{v}^{(i)}}_{i=1...N}$,
K : $intrinsics$,
P: set of perturbation transformations
Result: $(E(.;\epsilon), \mathbf{T}_l^c)$: trained encoder network and link-to-camera Transformation
for $k$ in number of iterations do
    Sample perturbation $\mathbf{p}^{(j)} \in \mathbf{P}$;
    $\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)} \leftarrow \sigma(\mathbf{x}^{(i)}, \mathbf{d}^{(i)}, \mathbf{K}, \mathbf{p}^{(j)})$ ;      // $\sigma(.)$ geometric transformation on image
    $\hat{\mathbf{v}}_l(\mathbf{x}^{(i)}) \leftarrow E(\mathbf{x}^{(i)}; \epsilon)$;
    $\hat{\mathbf{v}}_l(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}) \leftarrow E(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}; \epsilon)$;
    $\hat{\mathbf{v}}_c(\mathbf{x}^{(i)}) \leftarrow \pi(\hat{\mathbf{v}}_l(\mathbf{x}^{(i)}), \mathbf{T}_l^c)$ ;      // $\pi(.)$ Local transformation on pose
    $\hat{\mathbf{v}}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)})) \leftarrow \pi(\hat{\mathbf{v}}_l(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)}), \mathbf{T}_l^c)$;
    $\mathcal{L}_l(\epsilon; \mathcal{D}) = \sum_{(\mathbf{x}^{(i)}, \mathbf{v}_l^{(i)}) \in \mathcal{D}} \|\hat{\mathbf{v}}_l(\mathbf{x}^{(i)}) - \mathbf{v}_l^{(i)}\|_2 + \mathcal{L}_{mixup}(\xi, \zeta; \mathcal{D})$ ;
    $\mathcal{L}_c(\mathbf{T}_l^c) \leftarrow \|\pi(\hat{\mathbf{v}}_c(\mathbf{x}^{(i)}), \mathbf{p}^{(j)}) - \hat{\mathbf{v}}_c(\mathbf{x}^{(i)} \circ \mathbf{p}^{(j)})\|_2$ ;
    update $\epsilon$ based on $\mathcal{L}_l$ ;
    update $\mathbf{T}_l^c$ based on $\mathcal{L}_c$ ;
end
```

Algorithm 3: DETECT camera calibration



(a) Transformation misalignment **t** of translation in x-component can be identified by choosing a perturbation transformation $\mathbf{p}^{(j)}$ with rotational component along z-axis, and comparing pose of geometrically transformed image $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})} \circ \mathbf{p}^{(j)})$ and direct local pose transformation $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})}) \circ \mathbf{p}^{(j)})$.



(b) Transformation misalignment **t** of rotation in z-component can be identified by choosing a perturbation transformation $\mathbf{p}^{(j)}$ with translational component along x-axis, and comparing pose of geometrically transformed image $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})} \circ \mathbf{p}^{(j)})$ and direct local pose transformation $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})}) \circ \mathbf{p}^{(j)})$.

Figure 3.2.: Illustrations demonstrating pose disparity resulting from misalignment of the true and predicted camera pose. Axes with square markers indicate true camera pose $\mathbf{v}_c(\mathbf{x}^{(\mathbf{i})})$, from which an image $\mathbf{x}^{(\mathbf{i})}$ is take, and axes with circular markers indicate predicted pose $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})})$ corresponding to the image $\mathbf{x}^{(\mathbf{i})}$. **t** is the transformation vector indicating misalignment between the true and predicted pose, **p** is a chosen perturbation transformation, $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})} \circ \mathbf{p}^{(j)})$ is pose prediction of geometrically transformed image (yellow), and $\hat{\mathbf{v}}_c(\mathbf{x}^{(\mathbf{i})}) \circ \mathbf{p}^{(j)})$ is local pose transformation (green) of predicted camera pose.

4 Experiments

In this chapter we organize experiments to evaluate the performance of DETECT algorithm introduced in Chapter 3. In Experiment 4.1 we evaluate the performance of a traditional AE against an encoder-decoder pair trained separately, to examine how well they are able to memorize the environment. In Section 4.2 we examine the performance of an encoder network with geometrically transformed images; evaluations are made with inclusion of mixup loss and image border clipping. In Section 4.3 we experiment with the DETECT algorithm with calibration tests restricted to 1 and 2 DoF. In Section 4.4 we perform a grid search around the true calibration parameters and examine the loss landscape.
All experiments were performed using a custom dataset, rendered with Blender3D, an open source 3D animation package. The dataset consists of RGB-D images, along with link poses, which are at a constant transformation from the true camera pose. In order to simplify the problem, robot geometry was excluded from the renders to omit any occlusions in camera view caused by robot links. Pose measurements in the dataset were only varied in position along the x and y-axis, and in rotation, along the z-axis; the restricted pose was sufficient to evaluate a calibration translation along the x-axis, and rotation along the z-axis.The dataset environment was setup with a camera pose at a translational offset of $+0.321m$ and an orientation offset of $+19°$ along x and z-axis, from the recorded link pose, respectively.

4.1 Environment Memorization

DNN have proved to effectively solve many complex learning problems, however, if the number of model parameters significantly exceeds the size of dataset, it is quite possible for the model to overfit the dataset [20]. In our experiments, the dataset is of a single environment, therefore it is possible that the network may not generalize well. We therefore design this experiment to examine if the model is able to find a generalized solution, rather than an overfit on the training data. In the experiment, we hold out data from certain regions of the environment and attempt to reconstruct using information from neighbouring regions. We evaluate an AE, which learns a latent code based on visual features of the image, and an Encoder-Decoder pair trained independently, which learn mapping between image and pose.

4.1.1 Experiment Setup

In this experiment, RGB images of dimensions 128×128 pixels were used, with pixel intensity of each channel normalized from $[0,255]$ to $[-1,1]$. AE and Encoder-Decoder pair used the same architecture, as shown in Appendix C, with network hyperparameters described in appendix D. The bottleneck layer for both models was set to output a vector of dimension 4; in case of the AE, the bottleneck output would be learnt by the model itself, whereas for the Encoder-Decoder pair, the first two components of the vector represents position along the x and y-axis, respectively, and the latter two represent rotational component along the z-axis, as described in Section 3.1. Data in range $[-5°,5°]$, $[-50°,-40°]$ and $[-165°,-155°]$ along the z-axis is held out from the networks to check for generalizability. The AE is optimized using a loss function:

$$\mathcal{L}_{AE}(\epsilon,\eta;\mathcal{D}) = \sum_{(x_i)\in\mathcal{D}} \ell(\hat{x}_i, x_i),$$

where ϵ and η are encoder and decoder network parameters, respectively, ℓ is an appropriate loss metric, like MSE, x_i represents the ith image from the dataset $\mathcal{D}$, and $\hat{x}_i$ is the image prediction corresponding to the ith sample. In the second case, the Encoder-Decoder pair is optimized using separate loss functions $\mathcal{L}_{enc}(\epsilon;\mathcal{D})$ and $\mathcal{L}_{dec}(\eta;\mathcal{D})$, respectively, defined as:

$$\mathcal{L}_{enc}(\epsilon;\mathcal{D}) = \sum_{(x_i,v_i)\in\mathcal{D}} \ell(\hat{v}_l(x_i), v_i), \tag{4.1}$$

$$\mathcal{L}_{dec}(\eta;\mathcal{D}) = \sum_{(x_i,v_i)\in\mathcal{D}} \ell(\hat{x}(v_i), x_i),$$

where ϵ and η are encoder and decoder network parameters, respectively, ℓ is an appropriate loss metric, like MSE, x_i and v_i represent the ith image and pose from the dataset $\mathcal{D}$, respectively, whereas $\hat{v}_l(x_i)$ and $\hat{x}(v_i)$ are link pose and image predictions made by encoder and decoder from ith image and pose, respectively.

4.1.2 Results

The performance of both models, trained on 150 epochs with data held out, as described in Section 4.1.1, is visualized in Figure 4.1. It can be observed that both models produce viable reconstructions from the hold out data and neither model overfits. However, the encoder-decoder pair produces results closer to the ground truth than the AE. We can attribute this enhanced performance of Encoder-Decoder pair to the fact that we enforce the true pose value at the bottleneck, therefore, latent space is structured according to the ground truth pose, whereas the AE has to estimate a bottleneck based on the image reconstruction loss at the end, which may not form as well a structured latent space, and may lead to slightly larger inaccuracies in viewpoint pose.

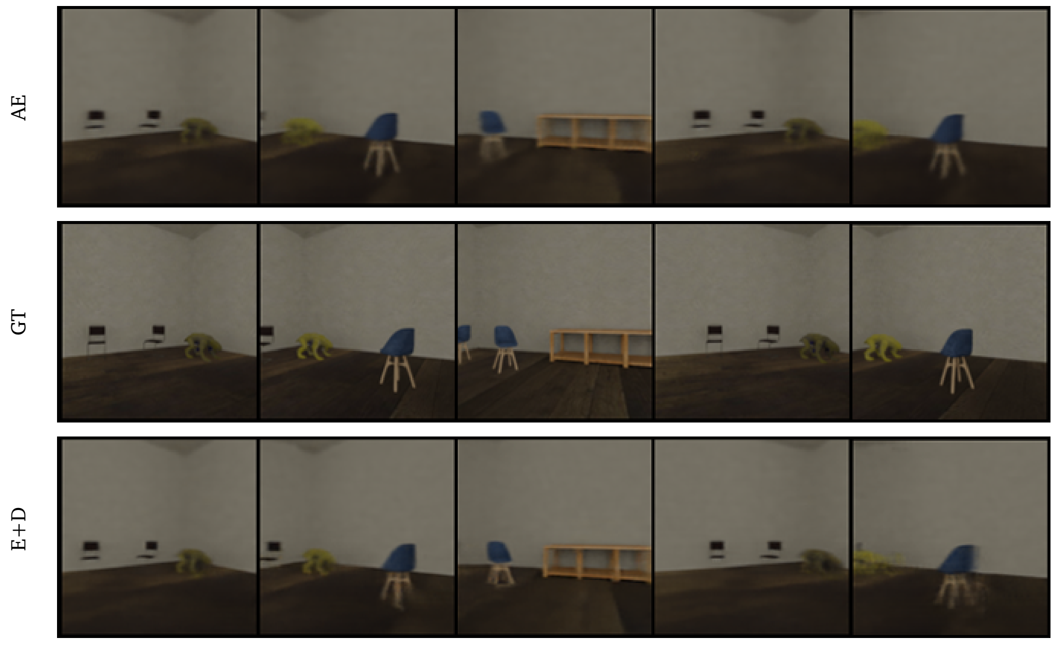


Figure 4.1.: Comparison of image reconstruction between AE and Encoder-Decoder trained independently.

4.2 Evaluation on Geometrically Transformed Images

This experiment is designed with the intention of testing the robustness of encoder network. Geometrically transforming an image leaves empty pixels around the borders, which may keep some neurons from firing that may be essential in determining the correct link pose. Ideally, we would like to have a model that is moderately dependent on all features in the image, so even if a few key elements are missing from the image borders, the network is still able to make a decent prediction. In this experiment we evaluate the performance of our encoder against a mixup variant, and another encoder trained on clipped input borders, along with its mixup variant.

4.2.1 Experiment Setup

In this experiment, we normalize channel intensities from $[0, 255]$ to $[-1, 1]$, use the encoder architecture as described in Appendix C, with hyperparameters described in Appendix D. The output size of each encoder is a vector of dimension 4, where the first two components of the vector represents position along the x and y-axis, respectively, and the latter two represent rotational component along the z-axis, as described in Section 3.1. Each network is trained on the complete dataset, with no hold out data. The unclipped variants of the network are input images of dimensions 128×128 pixels, while the clipped variants are fed the same center region of 72×72 pixels of the same image, i.e. the 28 pixels from each border is clipped. The mixup variants use an additional mixup loss as described in Section 2.2, besides the basic pose reconstruction loss from Equation 4.1.

4.2.2 Results

The validation results of each model variant are shown in Figure 4.2; these results only depict performance on untransformed samples from the dataset, since it would not be possible to acquire the correct link pose for a geometrically transformed image, unless the true camera calibration transformation is known. A qualitative comparison of image reconstruction from geometrically transformed input is shown in Figure 4.3 and 4.4, where the maximum geometric rotation was set at 12° along z-axis, to ensure that region inside the clipped image does not contain any empty pixels. It is evident from the plots in Figure 4.2 that clipped variants of the model lose some accuracy on the untransformed data samples, compared to the unclipped variants. However, as can be seen in Figure 4.4 they perform far better on geometrically transformed images. This signifies that empty pixels have a major impact on the decision boundary of a pose prediction. It is also worth noting that even though mixup loss does not improve the accuracy on untransformed samples, it does have a major impact on the robustness of the system, as can be seen in Figure 4.3, where although the predicted poses are not very accurate, they are a lot closer to the true poses, compared to the variant without mixup.

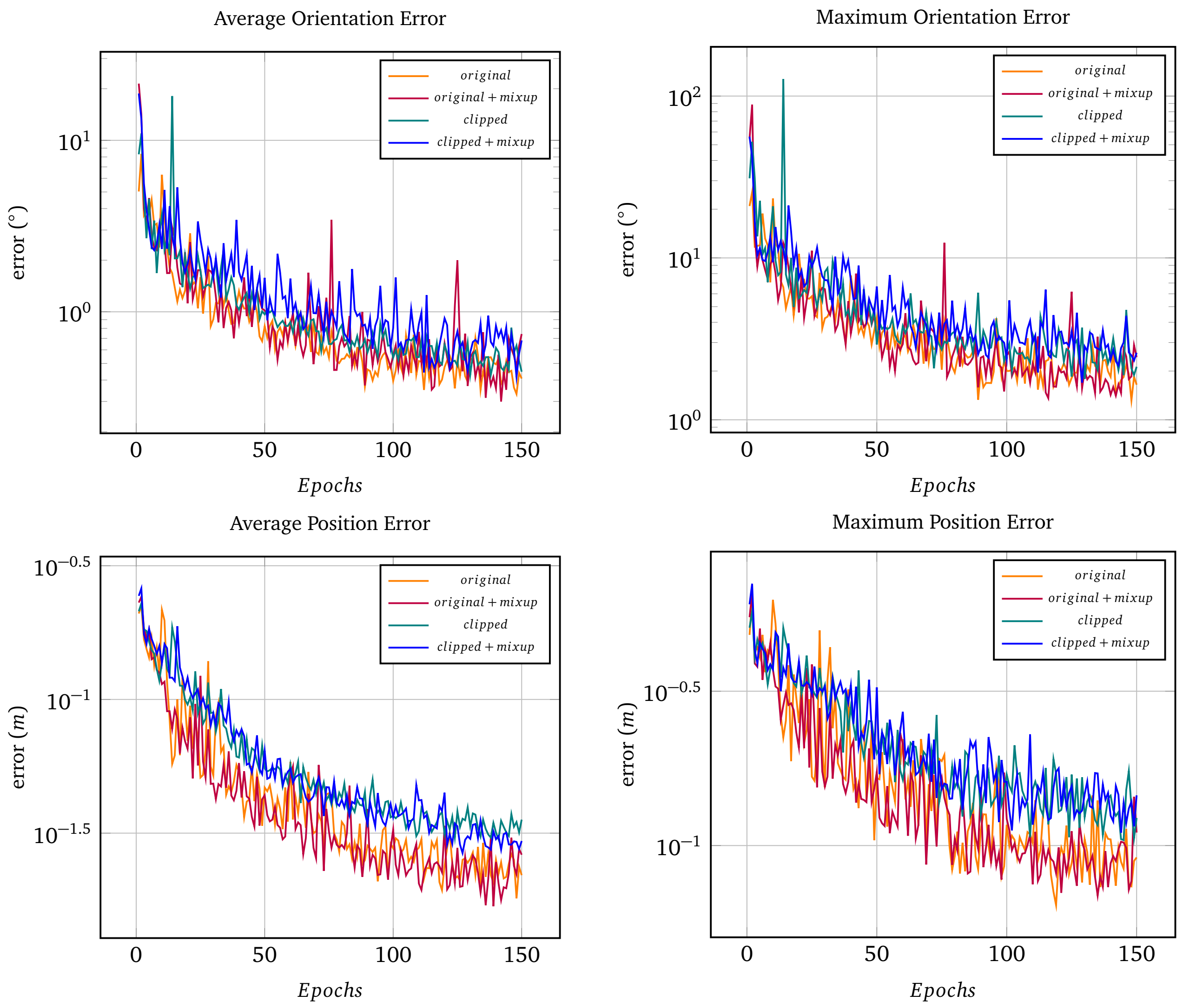


Figure 4.2.: Results of link pose prediction error computed on untransformed images. The plots show that unclipped variants of the model have a lower prediction error for both orientation and position predictions. Mixup on the other hand, does not seem to have any notable impact on prediction error of untransformed images.

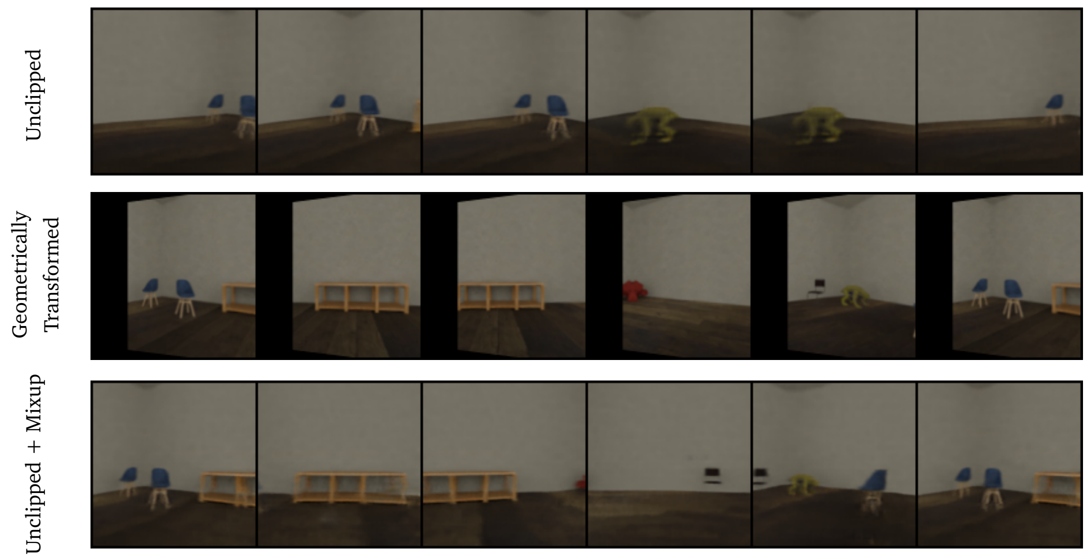


Figure 4.3.: Ouput Images produced by decoding pose predictions from geometrically transformed images, using unclipped variants of the encoder model. The original unclipped variant of the model is unable to make correct predictions, if fed with geometrically transformed images. The mixup variant on the other hand, is able to isolate a pose in the general vicinity of the true link pose, however, it is still off by a significant margin.

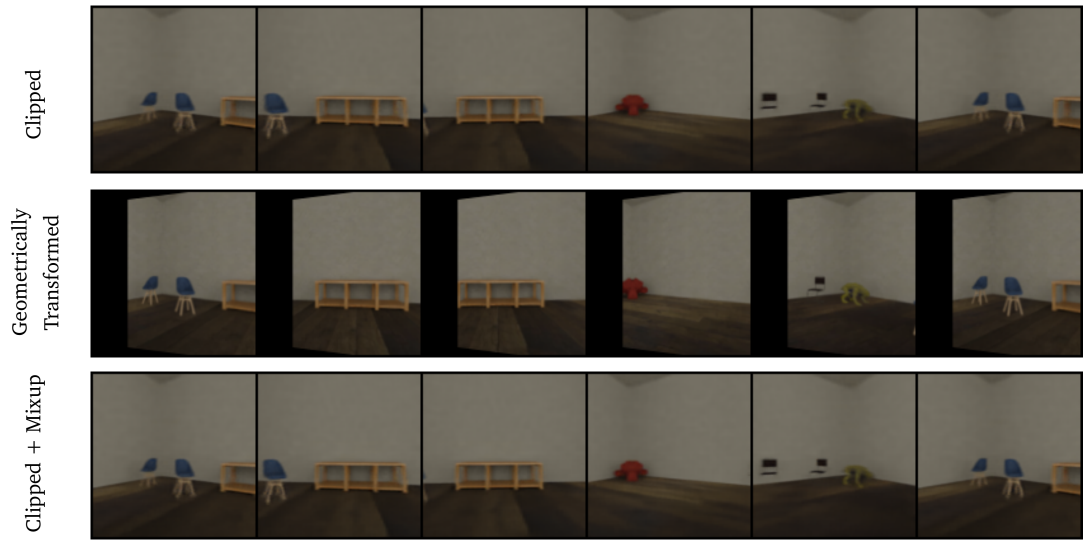


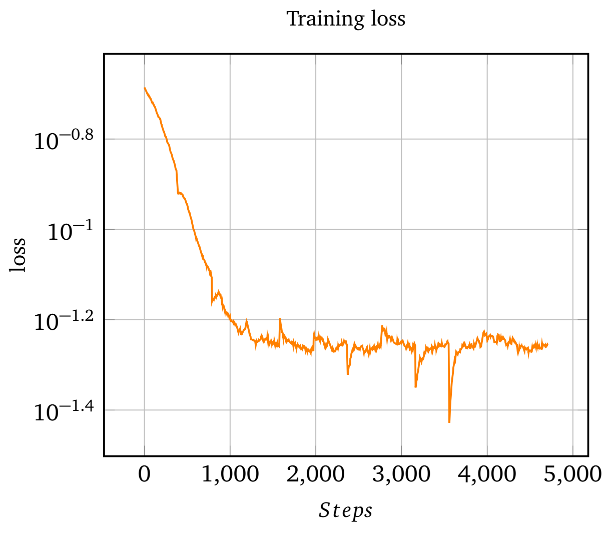
Figure 4.4.: Ouput Images produced by decoding pose predictions from geometrically transformed images, using clipped variants of the encoder model. Both clipped variants, with and without the mixup perform equally well to isolate the pose for geometrically transformed images.

4.3 Evaluation of End-to-End Calibration

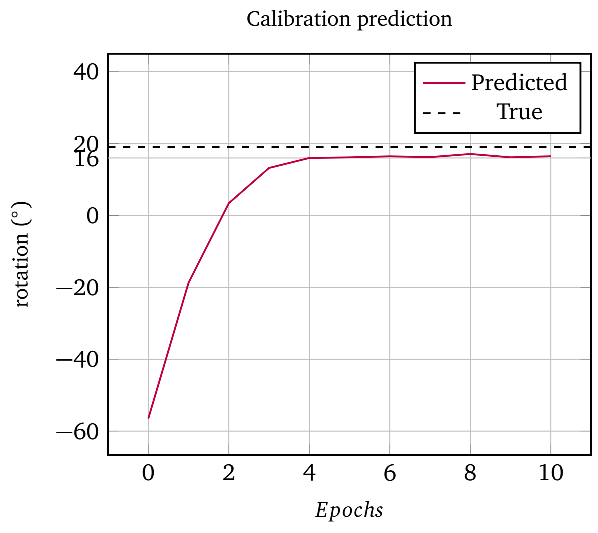
This experiment is aimed at evaluating the performance of end-to-end calibration algorithm presented in Section 3. As shown in Figure 3.2, miscalibration of z rotation axis and x translation axis, introduces positional disparity only in the xy plane, therefore, it makes sense to first evaluate the performance within this plane. The results of these evaluations could also be extended, to project performance of calibration on other rotation-translation axis pairs. In the experiment we utilise a pretrained encoder from Section 4.2, with the best performance, i.e. the clipped input variant, with mixup loss. We then evaluate the performane of our algorithm with 1 and 2 DoF, in Section 4.3.1 and 4.3.2.

4.3.1 Calibration with 1 DoF

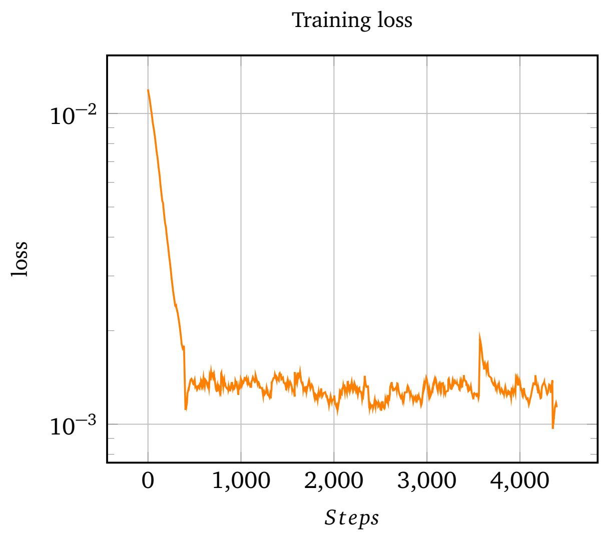
We start with the simplest case; the calibration prediction model from Figure 3.1 is only allowed to predict one calibration transformation parameter, the other parameters are fixed to the original calibration values of the dataset. The model was allowed to train until the calibration loss plateaued, and the results can be visualized in Figure 4.5, where it can be observed that even for 1 DoF the calibration predictions do not converge to the true value; the offset in calibration prediction can be attributed to inaccuracies in image reconstructions from geometric transformation. Once an image undergoes geometric transformation beyond a certain value, side planes of geometry come into camera view, however, since that information is occluded in the original image and its depth map, the exact reconstruction of the side planes is not possible, and any approximations made, inject error into the transformed image.



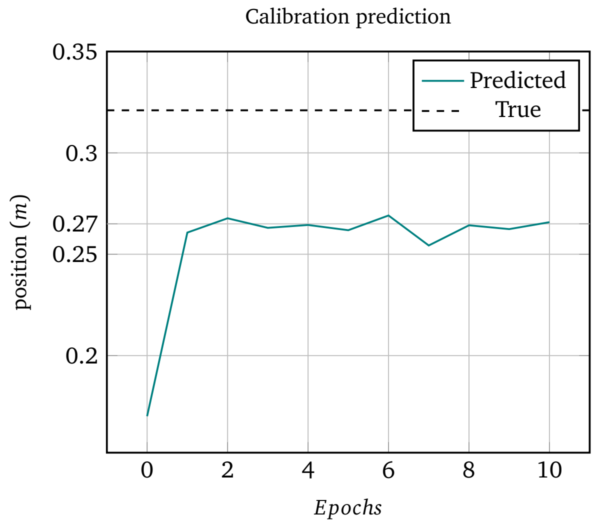
(a)



(b)



(c)



(d)

Figure 4.5.: End-to-end calibration evaluation with 1 DoF. 4.5a shows training loss and 4.5b shows prediction if calibration DoF is rotation along z-axis, 4.5c shows training loss and 4.5d shows prediction if calibration DoF is translation along x-axis.

In the second experiment we allow calibration prediction model from Figure 3.1 to predict two calibration transformation parameters, i.e. the translation along x-axis and rotation along z-axis, the other parameters are fixed to the original calibration values of the dataset. The model was trained until the calibration loss plateaued, results for which are presented in Figure 4.6. It can be observed that both calibration predictions are different from the 1DoF case, presented in Section 4.3.1. It is also worth noting that rotational component of the prediction is more stable than the translational component; by comparing loss functions in Figure 4.5a and 4.5c, we find that the pose disparity resulting from miscalibration of rotational component is manyfolds greater than the translational component, which is why the calibration model first optimizes for the rotational component and then the translational component, which may lead to a non-optimal minima for the translational component.

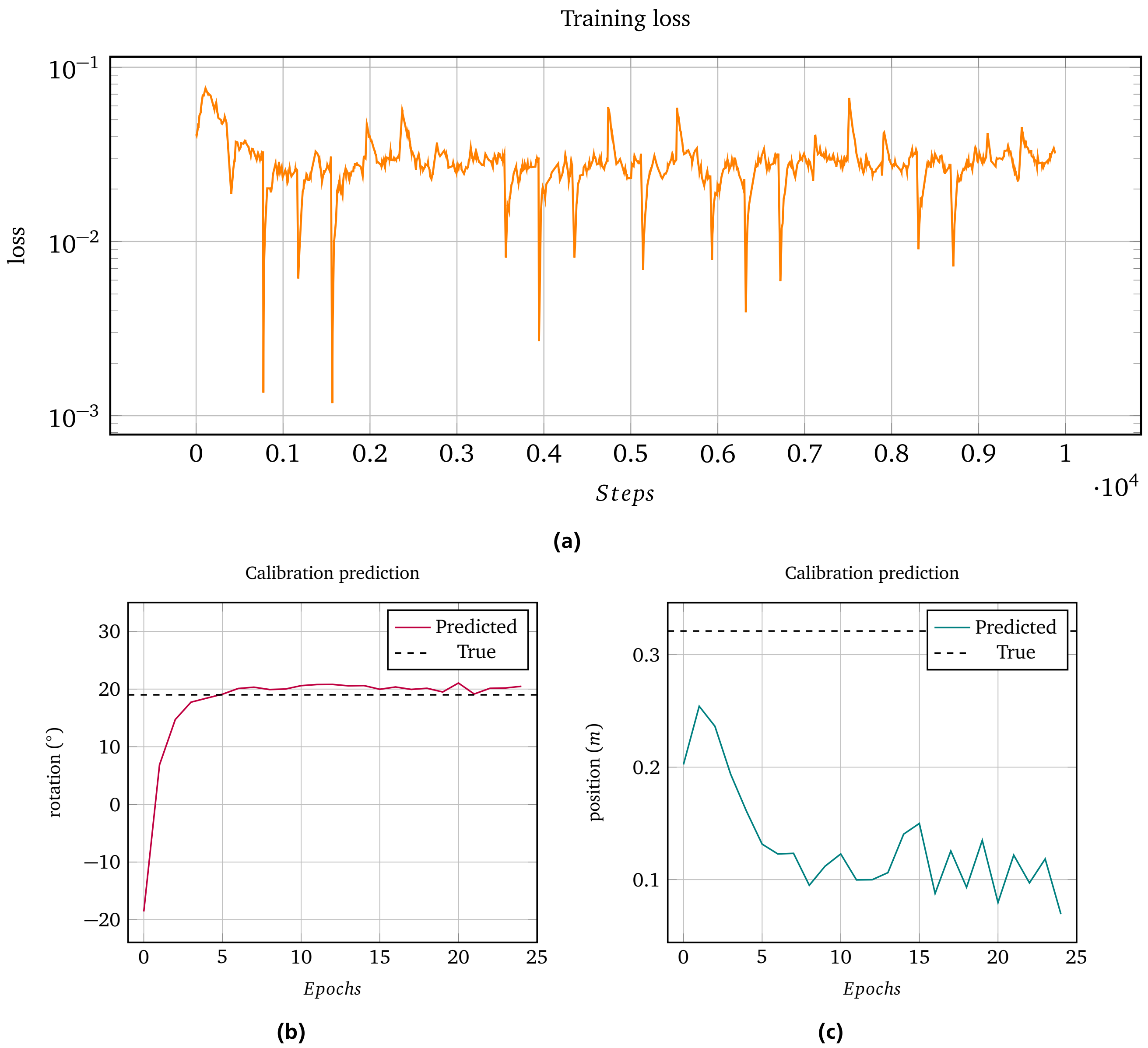


Figure 4.6.: End-to-end calibration evaluation with 1 DoF. 4.5a shows training loss and 4.5b shows prediction if calibration DoF is rotation along z-axis, 4.5c shows training loss and 4.5d shows prediction if calibration DoF is translation along x-axis.

4.4 Evaluation of Loss Landscape

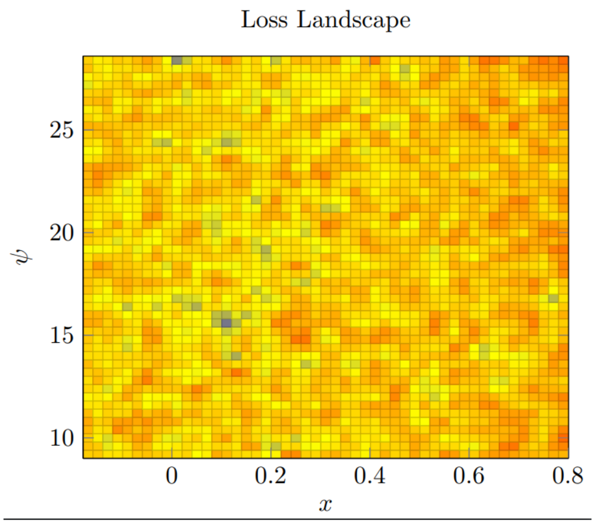
In this experiment we perform a grid search around the optimal calibration parameters with 2 DoF, to inspect the loss landscape of the proposed algorithm. The loss landscape provides a more clear insight of the optimizer's behaviour, and may help in rooting out the cause of inconsistent convergence points for systems with different DoF.

4.4.1 Experiment Setup

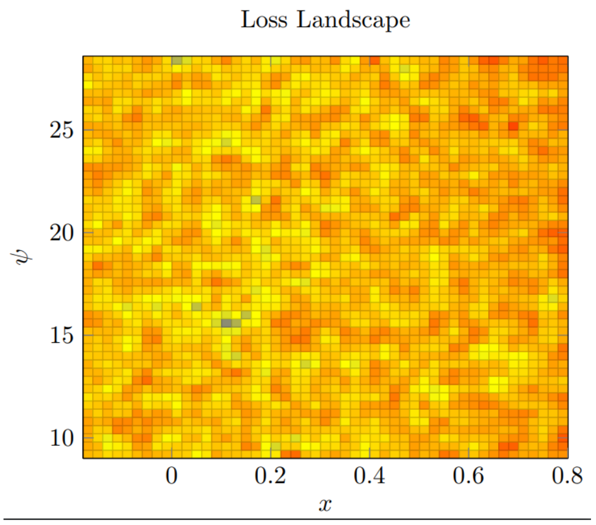
In this experiment we normalize channel intensities from [0, 255] to [-1, 1], the model used is the same as for experiment in Section 4.3.2, however, instead of the acquiring calibration transformation from calibration prediction network, we perform a grid search on translation in range $[-0.5m, 0.5m]$ around the expected optimal value of $0.321m$ along x-axis, and rotation in range $[-10°, 10°]$ around the expected optimal value along z-axis; the remaining calibration parameters were fixed to their ground truth values. The grid search was conducted twice, once for a partially trained model with an average position error of $0.03m$ and average orientation error of $0.56°$, and again for a fully trained model with average position error of $0.02m$ and average orientation error of $0.35°$; the intention was to inspect the evolution of loss landscape as the accuracy of pose predictions improve.

4.4.2 Results

The results from grid search can be seen in Figure 4.7, where it is evident that for both models, the loss landscape is quite complex, with numerous valleys, randomly scattered all around the search region. The optimal calibration point with the proposed algorithm can be found at a z-axis rotation of $15.6°$ and x-axis position of $0.12m$, however, these values are far from the true calibration parameters of $19°$ and $0.321m$, respectively. We attributte the shift in optimal calibration parameters point to imperfect results of geometric transformation. It can also be observed that the landscape does not seem to show a systematic global minima, and therefore depending on the starting point and DoF, the calibration model might converge to a different solution every time. By comparing the landscape of partially trained model vs fully trained model, we find that the partially trained model, with lower accuracy, has a greater number of local minima, with values closer to the optimal minima, which may falsely be registered as the optimal point in the landscape. It can therefore be deduced that the proposed algorithm is also sensitive to prediction accuracy, which further complicates the calibration process.



(a)



(b)

Figure 4.7.: Loss landscape as a result of grid search around true calibration parameters using a partially trained model in 4.7a, and fully converged model 4.7b. The general landscape in both cases are similar; however, the fully trained model shows higher loss values at points that do not represent the optimal calibration parameters.

5 Discussion

In this thesis we propose an algorithm for end-to-end hand-eye calibration using deep learning, as a possible means of reducing errors due to the human involvement in traditional calibration techniques. We showed that DNNs are effective in learning an environment given pose and corresponding image samples from the environment. In a comparison between Auto-Encoders (AE) and separately trained encoder-decoder pair, we found that the encoder-decoder pair exhibited superior performance in capturing the environment. Since the bottleneck layer of the encoder-decoder pair has an extra training signal of ground truth pose, it is able to arrange data points in a more organized manner. With more training data the prediction accuracy of these models can further be enhanced.
Upon evaluation, we found that although standard encoder architectures were good at memorizing the relationship between images and pose of an environment, they were not very robust. Missing pixel information in the query image could severely throw the network off of the actual view pose, which was an undesirable response. In order to implement our calibration algorithm we required the model to be resistant towards missing data regions, especially, since our method required pose predictions of geometrically transformed images, which invariably left missing pixel information around the image borders. We showed that introducing a mixup [17] signal in the training process improved the system robustness, however, it was still insufficient to achieve the required accuracy standards for our algorithm. We found that clipping image borders, on top of the mixup signal, proved to be a more effective solution.
Evaluation of the DETECT algorithm was a challenging task due to volatile response of the system, depending on the starting conditions and DoF assigned to the calibration network, predicted transformations converged to different solutions. A grid search around the true calibration parameters revealed that our proposed loss function produced a very complex loss landscape, which made it difficult for the optimizer to consistently converge to a common solution for different DoF. This is a major fault in the designed loss function, since it does not offer a loss landscape that steadily tapers towards the global minima, gradient descent will get stuck in any one of the landscape's local minima, depending on the starting conditions and allowed DoF. We also found that geometric transformations were also not accurate enough for predicting a good calibration, the optimal point in the loss landscape was shifted away from the true calibration point, as a result of these inaccuracies.
This thesis laid the foundation of a possible end-to-end hand-eye calibration technique using deep learning. The proposed algorithm was implemented and evaluated on a restricted dataset, with minimal DoF required to assess the algorithm's performance. It was shown that the algorithm performed well in memorizing the environment, however, the calibration portion needs to be developed further, before it can be deployed on actual robotic systems.

5.1 Future Work

The calibration portion of the proposed algorithm offers a lot of opportunities for further research. In the current system, inaccuracies in the geometrically transformed images leads to bad calibration parameters; it would be interesting to see if we can remove dependence of our system from geometric transformation. A possible solution might be to apply known transformation to pose in the bottleneck layer and examine the optical flow of the decoded images, and check for consistency between the pose transformations and optic flow. Although such a system might also suffer from complex loss landscape, however, such an arrangement might correct the offset between optimal and true calibration parameters.
It is also worth investigating how prior knowledge might improve the calibration predictions; if we feed a rough estimate and a search range to the current system, it may be possible to locate true calibration parameters. Since the starting point would be closer to the true calibration parameters, it would be highly probable that the calibration model converges to the correct solution.
In our experiments we used a synthetic dataset, with consistent lighting conditions, however, in a real data collection session, the lighting might vary, and the camera view might be occluded by the robot links; it would be interesting to evaluate the environment memorization capability with augmented data. We have seen that missing pixels severely hampers the pose prediction accuracy in our system,data augmentation might help enhance the model robustness.

Bibliography

[1] C. Nissler, Z. Márton, H. Kisner, U. Thomas, and R. Triebel, "A method for hand-eye and camera-to-camera calibration for limited fields of view," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5868–5873, Sep. 2017.

[2] T. Chin, R. Ding, and D. Marculescu, "Adascale: Towards real-time video object detection using adaptive scaling," *CoRR*, vol. abs/1902.02910, 2019.

[3] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[4] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: real-time instance segmentation," *CoRR*, vol. abs/1904.02689, 2019.

[5] M. Brown and D. G. Lowe, "Unsupervised 3d object recognition and reconstruction in unordered datasets.," in *3DIM*, vol. 5, pp. 56–63, 2005.

[6] C. Li, X. Peng, S. Zhang, J. Li, and L. Wang, "Hierarchical attention networks for knowledge base completion via joint adversarial training," *CoRR*, vol. abs/1810.06033, 2018.

[7] E. U. Samani, W. Guo, and A. G. Banerjee, "Deep learning-based semantic segmentation of microscale objects," *arXiv preprint arXiv:1907.03576*, 2019.

[8] R. Herzig, M. Raboh, G. Chechik, J. Berant, and A. Globerson, "Mapping images to scene graphs with permutation invariant structured prediction," in *Advances in Neural Information Processing Systems*, pp. 7211–7221, 2018.

[9] B. Llanas, S. Lantarón, and F. J. Sáinz, "Constructive approximation of discontinuous functions by neural networks," *Neural Processing Letters*, vol. 27, no. 3, pp. 209–226, 2008.

[10] A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," *CoRR*, vol. abs/1704.00390, 2017.

[11] A. Kendall, M. Grimes, and R. Cipolla, "Convolutional networks for real-time 6-dof camera relocalization," *CoRR*, vol. abs/1505.07427, 2015.

[12] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," *CoRR*, vol. abs/1711.00199, 2017.

[13] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, "Demon: Depth and motion network for learning monocular stereo," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5038–5047, 2017.

[14] G. Gao, M. Lauri, J. Zhang, and S. Frintrop, "Occlusion resistant object rotation regression from point cloud segments," *CoRR*, vol. abs/1808.05498, 2018.

[15] T. Do, M. Cai, T. Pham, and I. D. Reid, "Deep-6dpose: Recovering 6d object pose from a single RGB image," *CoRR*, vol. abs/1802.10367, 2018.

[16] A. Csiszar, J. Eilers, and A. Verl, "On solving the inverse kinematics problem using neural networks," in *2017 24th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, pp. 1–6, Nov 2017.

[17] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *ICML*, 2018.

[18] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *CoRR*, vol. abs/1710.09412, 2017.

[19] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, “On the continuity of rotation representations in neural networks,” *CoRR*, vol. abs/1812.07035, 2018.

[20] Z. J. Xu, Y. Zhang, and Y. Xiao, “Training behavior of deep neural network in frequency domain,” *CoRR*, vol. abs/1807.01251, 2018.

[21] S. A. Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, *et al.*, “Neural scene representation and rendering,” *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[22] K. Ridgeway, J. Snell, B. Roads, R. S. Zemel, and M. C. Mozer, “Learning to generate images with perceptual similarity metrics,” *CoRR*, vol. abs/1511.06409, 2015.

[23] Z. Zhang, T. He, H. Zhang, Z. Zhang, J. Xie, and M. Li, “Bag of freebies for training object detection neural networks,” *CoRR*, vol. abs/1902.04103, 2019.

[24] N. Passalis and A. Tefas, “Improving face pose estimation using long-term temporal averaging for stochastic optimization,” in *International Conference on Engineering Applications of Neural Networks*, pp. 194–204, Springer, 2017.

[25] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

[26] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015.

[27] A. Kendall and Y. Gal, “What uncertainties do we need in bayesian deep learning for computer vision?,” *CoRR*, vol. abs/1703.04977, 2017.

[28] T. Ucicr, “Feature-based image metamorphosis,” *Computer graphics*, vol. 26, p. 2, 1992.

A Homogeneous Transformations ($\mathbb{R}^3$)

3D Transformation	d.o.f.	H	H
Translation	3	$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Rotation x-axis	1	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{R}_x & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Rotation y-axis	1	$\begin{bmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{R}_y & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Rotation z-axis	1	$\begin{bmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{R}_z & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Affinity	12	$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$
Projectivity	15	$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$	$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{t} \\ \boldsymbol{p}^T & 1/\lambda \end{bmatrix}$

B Evidence Lower Bound (ELBO) Derivation

$$\begin{aligned}
\log p(\mathbf{x}) &= E_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x})] && \text{(B.1)}\\
&= E_{q(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\right)\right] && \text{(B.2)}\\
&= E_{q(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\cdot\frac{q(\mathbf{z}|\mathbf{x})}{q(\mathbf{z}|\mathbf{x})}\right)\right] && \text{(B.3)}\\
&= E_{q(\mathbf{z}|\mathbf{x})}\left[\log\left(p(\mathbf{x}|\mathbf{z})\cdot\frac{1}{\frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})}}\cdot\frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right)\right] && \text{(B.4)}\\
&= E_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - E_{q(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z})}\right)\right] + E_{q(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{q(\mathbf{z}|\mathbf{x})}{p(\mathbf{z}|\mathbf{x})}\right)\right] && \text{(B.5)}\\
&= E_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})] + D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})] && \text{(B.6)}\\
\log p(\mathbf{x}) - \underbrace{(E_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})])}_{ELBO} &= \underbrace{D_{KL}[q(\mathbf{z}|\mathbf{x})||p(\mathbf{z}|\mathbf{x})]}_{error\ in\ approximation} && \text{(B.7)}
\end{aligned}$$

C Model Architecture

layer name	output size	encoder
conv1	128×128	7×7, 64, stride 2
conv2_x	64×64	3×3, max pool, stride 2
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	32×32	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
conv4_x	16×16	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv5_x	8×8	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
	1×1	average pool, 512-d fc
	1×1	4-d fc

Table C.1.: resnet based encoder network architecture

layer name	output size	decoder
	1×1	4-d fc
	1×1	4096-d fc
deconv1_x	16×16	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
deconv2_x	32×32	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
deconv3_x	64×64	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
deconv4_x	64×64	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
deconv5	128×128	3×3, stride 1

Table C.2.: resnet based decoder network architecture, where deconv layer corresponds to ConvTranspose2d in pytorch

D Model Hyperparameters

Model name	Optimizer	Learning rate	Epsilon	L2 penalty
encoder	Adam	$5e^{-3}$	$1e^{-8}$	0
decoder	Adam	$5e^{-3}$	$1e^{-8}$	0
AE	Adam	$5e^{-3}$	$1e^{-8}$	0
Calibration Predictor	Adam	$1e^{-3}$	$1e^{-8}$	0

Table D.1.: Model hyperparameters for encoder, decoder, AE, and calibration networks.