Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische Friedrich-Wilhelms-Universität Bonn

Bachelor Thesis

Learning Category-Level Coordinate System for 6D Object Pose Estimation

Author:
Patrick Lowin

First Examiner:
Prof. Dr. Sven Behnke

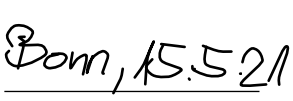
Second Examiner:
Priv.-Doz. Dr. Volker Steinhage

Advisor:
Arul Periyasamy

Date: May 15, 2021

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.



Place, Date

Signature

Abstract

In this thesis, we aim to learn a category-level coordinate system, which is jointly optimized with the task of category-level 6D pose estimation, using only a single RGB image and mesh information. We predict the translation by detecting the learned origin with a heatmap, which we then project back into 3D space with an estimated depth value. The rotation estimation is decoupled from the translation and split into two tasks. First, we predict a transformation to a canonical coordinate system, which we optimize with our CanonicalLoss. Secondly, we estimate the transformation from the shared space to the camera frame.

Due to the lack of high-quality data for category-level pose estimation, we generate realistic images using the Stillleben rendering pipeline. The synthetic scenes are heavily cluttered and complex in object interactions leading to heavy occlusions, common in many warehouse settings.

Experiments with our pipeline investigate the effects of the canonical coordinate system, the canonical loss, the performance of different rotation representations, and the generalization to unseen instances.

Contents

1 Introduction **1**

2 Fundamentals **3**

2.1 Neural Networks . . . 3
2.1.1 Artifical Neurons . . . 3
2.1.2 Multilayer Perceptrons . . . 3
2.1.3 Convolutional Neural Networks . . . 4
2.1.4 Optimizing Neural Networks . . . 5
2.1.5 Learning Rate Scheduler . . . 6
2.2 Region of Interest . . . 7
2.3 Heatmaps . . . 7
2.4 Coordinate Systems and Transformations . . . 8
2.5 3D Data Representations . . . 9
2.5.1 Euclidean-structured Data . . . 9
2.5.2 Non-Euclidean-structured Data . . . 10
2.6 RefineNet . . . 10
2.7 6D Pose estimation . . . 11
2.8 PoseCNN . . . 11
2.8.1 Rotation Regression . . . 12
2.8.2 Translation Regression . . . 12
2.8.3 Evaluation metric . . . 13
2.9 Continuity of Rotations Representations . . . 14
2.9.1 Continuous Representation . . . 15
2.9.2 Singular Value Decomposition for Deep Rotation Estimation 16
2.10 Feature Learning on Point Sets . . . 16

3 Related Work **19**

3.1 6D Pose Estimation . . . 19
3.1.1 Instance-Level 6D Pose Estimation . . . 19
3.1.2 Category-Level 6D Pose Estimation . . . 20
3.2 Semantic Segmentation . . . 20
3.3 Canonical Representations . . . 21

3.4 Training Data Generation 22

4 Our Approach **25**
4.1 Synthetic Dataset 25
4.1.1 Realistic Cluttered Scenes using Stillleben and DeepCPD . . 25
4.1.2 Limitations of Synthetic Data Generation 27
4.1.3 Canonical Coordinate System 27
4.1.4 Optimizing the Canonical Coordinate System 28
4.1.5 Adjusting Ground Truth Data 29
4.1.6 Regularization 30
4.2 Network Architecture 31
4.2.1 Canonical Rotation Estimator + Translation - CaRET . . . 31
4.3 Training Implementation 35

5 Experiments **37**
5.1 Experiments with CaRE 37
5.1.1 Comparison of Rotation Representations 37
5.1.2 Canonical Loss Weighting 38
5.1.3 Evaluation on Unknown Drills 39
5.2 Experiments with CaRET 40
5.2.1 Evaluation of the Canonical Module 40

6 Conclusion **53**

Appendices **55**

1 Introduction

In order to interact with its environment, a robot has to know the 6D pose, i.e., position and orientation, of the objects it wants to interact with. The task gets increasingly more complex, with the number of instances in a scene. Cluttered environments introduce large occlusions, which are common in warehouse settings, making it even more important to have algorithms that solve this task robustly. Due to their learning ability, deep neural networks can handle these conditions, whereas other methods like template matching methods struggle to get accurate results. A large portion of the research focuses on instance-level pose estimation, where the network learns to predict the poses for a set of objects. However, this approach does not scale well due to the vast number of unique objects, because each object instance would require its own output. For example, the YCB-Dataset[1] includes two clamps, and PoseCNN, therefore, predicts two rotations. However, one can solve this scaling issue by organizing objects of similar geometry into categories and learn a category-level pose estimation.

He Wang et al.[2] were the first to tackle this problem for hand-scale objects by learning a mapping from object image pixels to a representative of a specific category. They use the predicted correspondences and additional depth information to estimate the full metric 6D pose and size of the objects using a pose fitting method.

A canonical representation is vital to transfer knowledge to previously unseen instances. Therefore, objects need to be prealigned into a common coordinate system or have a canonical representative like [2] to perform pose estimation on a category level. This arbitrary coordinate system, might be suboptimal for the category.

This novel approach tries to account for all these aspects and jointly learns category-level pose estimation and a canonical coordinate system.

We build a two-stage pipeline, where the first stage focuses on object detection and translation, and the second stage on rotation. We use a RefineNet backbone to predict a category-level segmentation, the distance in the z-direction, and a heatmap that locates the object's origin. Knowing the object's distance and origin in the image plane, we can estimate the 3D translation by projecting the origin back into 3D space. In the second stage, we extract object features from a given

mesh with a PointNet and predict the canonical pose with our canonical module. We want the objects to be similarly orientated in the canonical coordinate system. We, therefore, introduce our CanonicalLoss that minimizes the shape differences for a category, aligning the object while not influencing the actual orientation. We then extract the object from the image and use slightly altered VGGs for each category to extract category-specific image features. A multilayer perceptron then predicts the final transformation from our canonical coordinate system to the camera frame from the image and mesh features.

We create our own data due to the lack of high-quality data sets for category-pose estimation. Data sets like the one used for NOCS [2] lack realistic lighting, clutter, a variety in object poses and occlusions. We use the Stillleben[3] rendering pipeline with the meshes from the DeepCPD data set[4] and create heavily cluttered scenes with occlusions of up to 40%. We provide a base train, validation, and test set but can increase the number of training images by generating them on the fly, allowing for life-long learning.

In the following section, we explain the concepts used for this thesis, followed by the ideas and architectures of other approaches. Finally, we present our pipeline in more detail and evaluate our approach in the experiments chapter.

We first evaluate the idea of higher-dimensional rotation representation, which performs better than traditional rotation representation due to their topological properties. Then, investigate our canonical module's effect on the quality of our transformation, training time, and the ability to transfer this knowledge to unknown instances. We evaluate our approach for instance-level pose estimation by testing on object instances present during training and category-level with previously unseen models.

2 Fundamentals

In this thesis, we want to learn a canonical representation for an object classes that we jointly optimize with the task of 6D pose estimation. Before introducing the pipeline, we review fundamentals from deep learning, computer vision, and robotics relevant to this thesis.

2.1 Neural Networks

2.1.1 Artifical Neurons

Artificial neural networks are inspired by their biological counterpart. Real neurons receive impulses over their dendrites which are then processed by the cell body to output an activation that travels to other neurons over the axons. This simplified explanation also describes artificial neurons, which take in multiple weighted inputs and outputs an activation. We calculate the activation of a neuron by summing over all weighted inputs and using a non-linear function.

$$o_j = \phi(\textstyle\sum_i w_{ij} x_i) + b_j$$

Here o_j denotes the output of the j-th neuron in the layer, w_{ij} the factor by which the input x_i gets multiplied, b_j the bias and ϕ is the activation function. A layer is fully connected (FC) if all neurons in the current layer connect to all other neurons in the next layer.

2.1.2 Multilayer Perceptrons

Multilayer perceptrons (MLP) consist of three types of layers. The first layer is called the input layer, followed by several hidden layers and an output layer. A layer consists of neurons with weighted connections to neurons in the previous layer, and computing the output from a given input is called a forward pass. The universal approximation theorem states that a neural network can approximate any continuous function given enough neurons[5, 6]. LeCunn et al.[7] explain that MLPs can distort the input space to make classes of data linearly separable, allowing them to learn complex functions. Linear classifiers separate their input

space into half-spaces with a hyperplane. However, tasks like object classification require the network to be invariant to changes of the input. Detecting a target in an image requires the network to be insensitive to background, illumination, and pose of the target.[7]

2.1.3 Convolutional Neural Networks

Convolutional neural networks (CNN) inspired by the visual cortex are designed to process array structured data like images. CNNs contain a series of convolutional layers which are organized into feature maps. A convolutional kernel K, a $N \times M$ array, slides over a feature map or image channel I, calculates the neighborhood's weighted sum and outputs to a unit in the next feature map. Here $*$ denotes the convolution operator:

$$(K * I)(i, j) = \sum_{m=1}^{M} \sum_{n=1}^{N} I(i - m, j - n)K(m, n)$$

The four main ideas behind CNNs are local connectivity, shared weights, pooling, and the use of multiple layers. In contrast to MLPs, where all pixels influence a neuron, a convolutional kernel only processes information in its receptive field. All units in feature maps share the weights of their kernel, solving two problems. First, it exploits the correlation of local groups of pixels. Secondly, it makes use of the fact that local statistics of images are invariant to location, allowing it to detect distinctive patterns that could appear anywhere in an image. The pooling layer then aggregates semantically similar features into one. Typically, we apply a max-pooling layerFigure 2.1 with stride $s > 1$. The stride specifies how many pixels we move until we apply the kernel again. Similar to a convolutional kernel, the max-pooling kernel slides over the image but only takes the maximum value in its area. Pooling reduces the dimension of the feature maps and creates an invariance to small shifts in the input space. A common practice is to increase the number of features while downsampling the image, allowing the network to learn an efficient representation of the input. The use of many layers is based on the idea that many low-level features form higher-level features.[7]
Early approaches created handcrafted features that humans could understand, but a CNN learns its own features, which we might not be able to comprehend. CNN's have been used for many tasks like object detection, semantic segmentation, face manipulation and image reconstruction. [8, 7]

Activation Functions After performing a convolution, an activation function calculates the activation of our neuron. The most popular being the Rectified

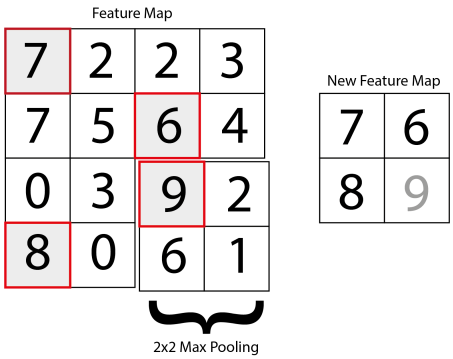


Figure 2.1: 2×2 Max pooling operation with stride=2. For each 2×2 patch, the max-pooling layer takes the maximum, downsampling the image.

Linear Unit (ReLU), a piecewise linear function that clips the negative part to zero and keeps the positive part. There is also the sigmoid function with a range between 0 and 1 that is fully differentiable. The advantage of ReLu over sigmoid is that it does not saturate for large activations. The sigmoid is saturated because it compresses the real numbers into the interval $I = [0, 1]$. The gradients become very small for large activations, causing the gradient to vanish. However, the ReLU suffers from the dying ReLU problem. A large negative bias may cause the neuron to output zero, which causes the neuron not to receive gradients. The Leaky ReLU solves this by having a slope for negative values, which adds a slight computational overhead but solves the dying ReLU problem[9, 10]

2.1.4 Optimizing Neural Networks

In this thesis, we train our network supervised and unsupervised. Supervised learning means that we have the corresponding ground truth to our input and optimize our network's parameters based on the loss calculated between ground truth and prediction. In unsupervised learning, we do not have labeled data. Nevertheless, we can impose certain constraints on the output. Training a network can be very time-consuming depending on the task's difficulty and the number of parameters. This section gives a brief introduction to the concepts used for optimizing a neural network.

Stochastic Gradient Descent

Many problems can be formulated as maximizing or minimizing a loss function, which are then iteratively optimized using gradient information. At each step, we calculate the gradient of our loss function w.r.t. to the whole data set and step in the gradient direction multiplied with the learning rate, which defines the step

size. After taking this step, we repeat the process until we find a local minimum. Calculating the gradient for the whole data set is not efficient and we therefore use stochastic gradient descent (SGD). The key difference to normal gradient descent is that we only approximate the actual gradient with random mini-batches of the whole dataset. This way, we do not need to save all the gradients and are thus able to train faster on larger datasets.
Libraries like PyTorch[11] are very efficient at calculating the updates for our network. PyTorch builds a computational graph where the nodes define the operations used at each step, e.g., multiplication and addition used for convolution. After the forward pass and calculating the loss, the backward pass computes the gradient using the chain rule and propagates the error back to the input[12, 13].
Different optimizers like AdaGrad, RMSProp, and Adam try to improve SGD. All of them use momentum, an aggregation of gradients, to calculate an update for our weights.[14, 15]

$$m_t = \beta m_{t-1} + (1 - \beta)\frac{\partial L}{\partial w_t}$$

Here m_t denotes the momentum at time step t, β is the weight of the past momentum and $\frac{\partial L}{\partial w_t}$ is the loss w.r.t. the weight w_t. The update to the weights is then computed with

$$w_{t+1} = w_t + \alpha m_t$$

2.1.5 Learning Rate Scheduler

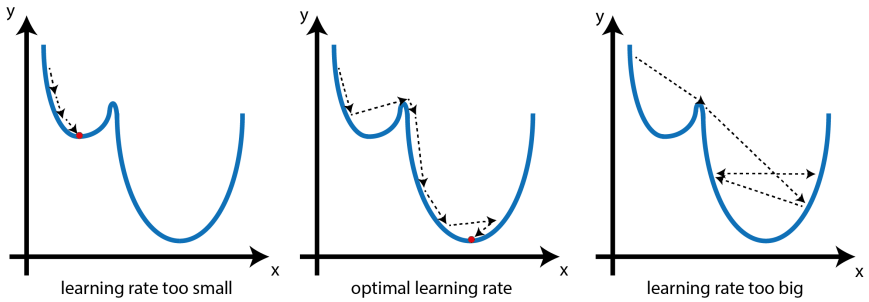


Figure 2.2: Comparison of learning rates. A small learning rate converges to a local minimum and does not explore the remaining space. The optimal learning rate explores more space and can find a better solution. A learning rate too big causes the network to oscillate and does not converge.

We initialize the optimizer with a fixed learning rate at the beginning of training. Finding the best learning rate is not an easy task Figure 2.2. Low learning rates lead to slow convergence, whereas larger learning rates cause the network to

diverge. Both can lead to suboptimal solutions. A learning rate scheduler adjusts the learning rate during training, leading to faster convergence and potentially to a better minimum. An easy example is monitoring the loss functions topology and reducing the learning rate on a plateau. During training, we perform a check if the loss has reduced in the last n epochs, and reduce the learning rate to find a local optimum if this is not the case. Another approach is decreasing the learning rate during training, which allows the network to take bigger steps to a minimum and then finetune its parameters. However, both of these schedulers have downsides. For example, smaller gradients of saddle points on the loss function cause slower convergence. Cyclic learning rate schedulers address this problem by cycling the learning rate between two boundaries. Smith[16] notes that the increase might have a short-term negative effect but beneficial effects in the long term. One intuitive upside is that traversal of saddle points is faster. However, a more practical reason is that the optimal learning rate lies between the two boundaries, and learning rates close to the optimum are used during training. The obvious downside is that the optimal learning rate has to be inside the bounds. Otherwise, we do not get this beneficial effect. Similar to one cycle in the previous example, CosineAnnealing sets the learning rate to a large value which is then rapidly decreased. The learning rate is then reset to a large value, which we refer to as warm restart and is supposed to restart the learning process. At each restart, the network converges more, and the parameters act as a better starting point[12, 16, 15].

2.2 Region of Interest

A region of interest (RoI) specifies a region in our image that contains valuable information. Some archiectures[10, 2] use 2D bounding boxes and others[1] semantic segmentation to predict RoIs. We can extract the regions from the image with a RoI pooling layer [17] and use them for further processing.

2.3 Heatmaps

We use heatmaps to detect interesting points in an image. Each pixel encodes the probability of this pixel being our point of interest. Newell et al.[18] use heatmaps to predict human joint location, and Law et al.[10] predict bounding boxes. We can encode a key point in a heatmap by placing a Gaussian bell over it. Our heatmap values increase the closer we get to the center, with a maximum value of 1. We can extract the keypoint by taking the $argmax$ of our heatmap, which

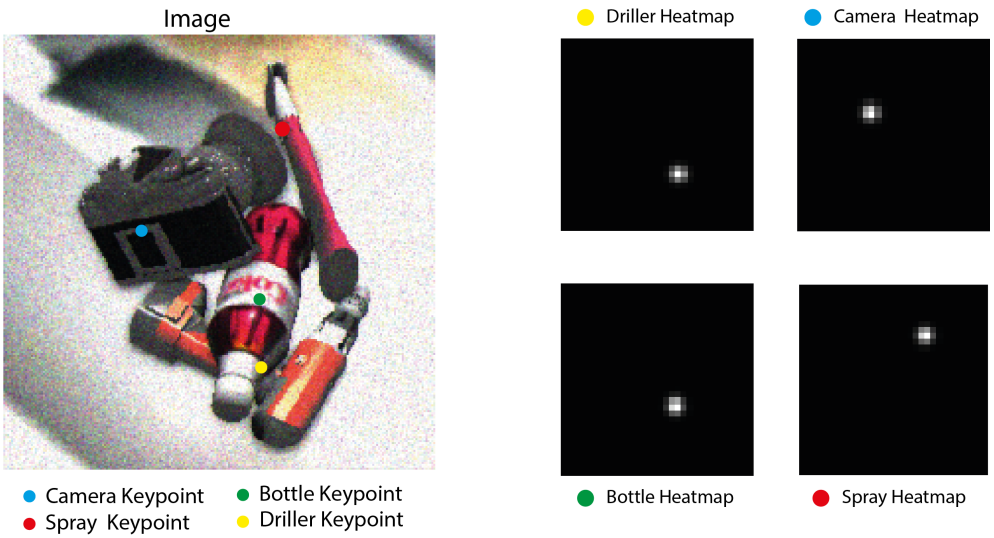


Figure 2.3: Heatmaps for origin detection. Each dot in the image represents an object center. The corresponding heatmap encodes the keypoint with an activity blob (Gaussian bell).

corresponds to the maximum indices. If we use a 2D Gaussian function e to encode our keypoint (k_x, k_y) then

$$(k_x, k_y) = argmax(e((k_x, k_y)))$$

2.4 Coordinate Systems and Transformations

To build a 3D model of our environment, we make use of rasterization-based rendering systems like OpenGL[19]. OpenGL or applications built on top of it like Stillleben[3] allow us to place objects in our world coordinate system and render an image of them. A coordinate system in 3D space consists of 3 axes $X, Y, Z \in \mathbb{R}^3$ and defines every point w.r.t the axes. We call the point $(0, 0, 0)$ the origin of the coordinate system. For example, the object coordinate system defines the position of points in the point cloud or vertices of the mesh. When placing an object in a scene, we move the origin with a translation vector $t \in \mathbb{R}^3$ to the desired location. We can also rotate objects around each axis with a rotation matrix $R \in SO(3)$. The special orthogonal group $SO(3)$ denotes all rotation matrices in 3 dimensions. Rotations can also be represented in four dimensions with quaternions or euler-angles defining the rotation around each axis. A transformation T combines R and t in a 4×4 matrix:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$$

We can switch coordinate systems by chaining transformations. For example, we can transform an object from its local coordinate system to the scene with T_O^W, the transformation from object-to-world coordinates. T_O^W also denotes the object pose in Stillleben and other renderers. To get the object pose in the camera frame, we multiply the transformations from world-to-camera and object-to-world: $T_O^C = T_W^C T_O^W$. Note that $T_W^C = T_C^{W-1}$ is the inverse of the camera pose in the scene.

2.5 3D Data Representations

3D data has a lot of different representations depending on the device that captured it. Eman Ahmed et al.[20] divide them into two classes: euclidean and non-euclidean data, which are illustrated in Figure 2.4

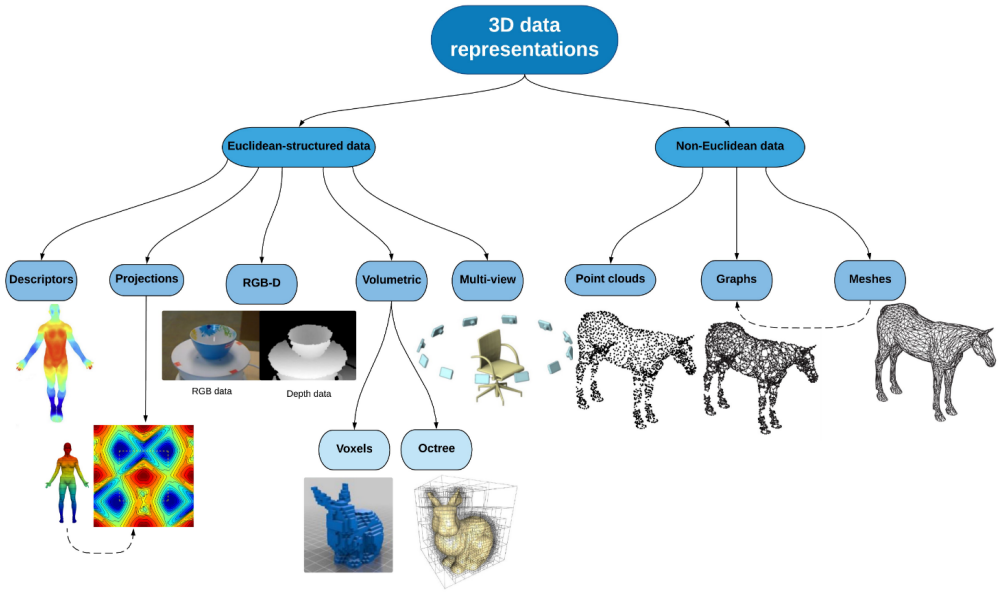


Figure 2.4: Overview of 3D data representations. Existing representations can be broadly categorized in euclidean and non-euclidean data. Image taken from [20].

2.5.1 Euclidean-structured Data

Euclidean data has an underlying grid structure that allows for a global parametrization and a common coordinate system. For example, an image has an underlying grid structure that allows us to define the position of every pixel with its coordinates. Depth information for images is easily accessible through sensors like the

Microsoft Kinect. The additional depth channel allows the refinement of an estimated pose from our network through the iterative closest point (ICP) algorithm. Another example of this category is volumetric data like voxel grids representing 3D geometry with a 3D grid. Each cube (voxel) in this grid encodes whether an object occupies it or not. Encoding both the presence and absence of objects is memory-inefficient, especially when dealing with high-resolution data.[20, 21]

2.5.2 Non-Euclidean-structured Data

The second category includes data representations without a vector space structure and global parametrization like point clouds and meshes. Point clouds are a set of points that approximates the geometry of an object. Point clouds are permutation invariant because permuting the points results in the same geometric features. Capturing point clouds is possible through structured light scanners like the Kinect1, ToF scanners like Kinect2, or passive methods like multi-view stereo. However, processing them in deep learning architectures is challenging due to the lack of structure. Meshes represent the geometry with a list of vertices and edges, which define which vertices are connected and form a face. They are commonly used in rendering applications to generate images and are also crucial for training 6D pose estimators. Ahmed et al. note that point clouds and meshes can be seen as both euclidean and non-euclidean data depending on the scale on which processing takes place.[20, 22]

2.6 RefineNet

Semantic segmentation is a dense classification problem and is crucial for object detection and image understanding. Deep CNNs like RefineNet achieve impressive accuracies at this task. RefineNet uses a multi-path refinement method to exploit features from multiple layers of abstraction. Their multi-path method uses four ResNet blocks to extract features from multiple scales of one image, which are then passed to their own RefineNet unit. Each RefineNet unit takes in additional features from previous units, which are then passed through a short-range residual convolution and fused together. Long-range residuals from the ResNets to the RefineNet units allow the later stages to use low- to mid-level features. The cascaded architecture can combine semantics from multiple levels to predict high-resolution segmentations and achieve state-of-the-art results.[23]

2.7 6D Pose estimation

Estimating the 6D pose is an essential task in robotics and scene understanding. It allows the robot to interact and manipulate its environment but is challenging due to varying appearances under occlusion and different lighting conditions. 6D pose estimation is the task of estimating an object's orientation and position using only a RGB or RGB-D image. More precisely, finding the rigid transformation from the object coordinate system to the camera frame.

$$M_{camera} = T_{object}^{camera} M_{object} \tag{2.1}$$

We define the pose in the camera coordinate system because of several reasons. First, the same image can correspond to different poses in the world coordinate frame. Secondly, estimating the depth is more straightforward and allows the use of depth cameras. [1, 2, 24]

2.8 PoseCNN

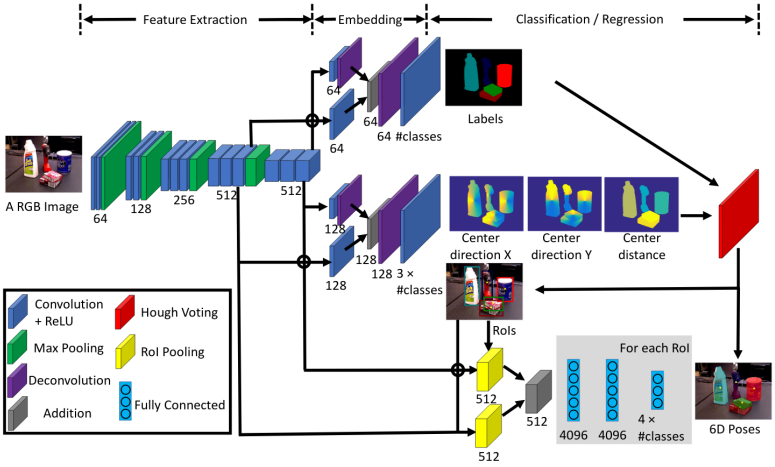


Figure 2.5: Architecture of PoseCNN. Image taken from [1]

PoseCNN is a state-of-the-art 6D object pose estimator. As can be seen in Figure 2.5, Xiang et al.[1] decouple the task of 6D pose estimation into predicting the 3D rotation and the 3D translation. PoseCNN predicts the semantic label, the depth, and a unit vector pointing to the object center for each image pixel. During a Hough Voting process, each vector votes for multiple pixels, and the pixel with the most votes is considered the center. Using object center, depth, and camera intrinsics, we can calculate the translation. The RoIs are extracted

from the segmentation and passed to a fully connected layer to predict the 3D rotation.[1]

2.8.1 Rotation Regression

PoseCNN introduces two novel loss functions to learn to regress the rotation component of the 6D object pose. The Pose Loss and the Shape Match Loss.

$$PLoss(q', q) = \frac{1}{2m} \sum_{x \in M} ||R(q')x - R(q)x||^2 \tag{2.2}$$

$$SLoss(q', q) = \frac{1}{2m} \sum_{x_1 \in M} min_{x_2 \in M} ||R(q')x_1 - R(q)x_2||^2 \tag{2.3}$$

Here M denotes all vertices on the mesh, m the number of elements in M, q' and q are the ground truth and predicted quaternion. Instead of regressing the quaternion, they rotate the object with the predicted and ground truth quaternion and calculate the average squared distance between corresponding points on the mesh for unsymmetric and the average squared distance to the nearest neighbor for symmetric objects [1].

2.8.2 Translation Regression

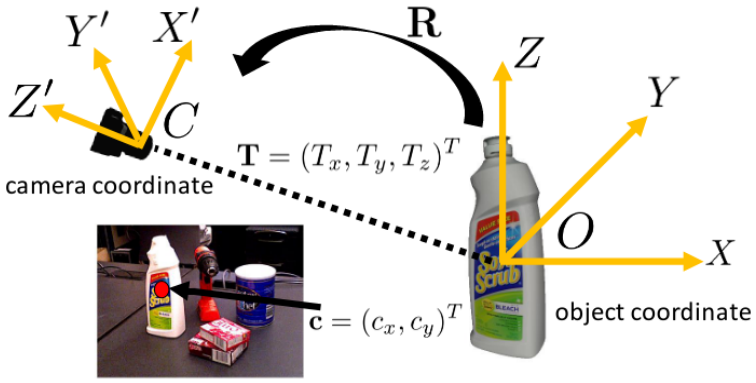


Figure 2.6: Illustration of the camera and object coordinate system. Knowing the center in the image and the distance in the z-direction, we are able to recover the translation T. Image taken from [1]

Xiang et al. propose a novel approach for predicting the translation of an object circumventing present problems when performing direct regression. One problem is that the network struggles to predict the correct translation when multiple objects

of the same category are present. Instead they predict the object center (c_x, c_y), shown in Figure 2.6 and the depth T_z and reconstruct the translation with known camera intrinsics:

$$\begin{bmatrix} \mathrm{c}_x \\ \mathrm{c}_y \end{bmatrix} = \begin{bmatrix} f_x \frac{T_x}{T_z} + p_x \\ f_y \frac{T_y}{T_z} + p_y \end{bmatrix} \tag{2.4}$$

Here f_x and f_y denote the focal length. They predict three output maps for each object instance. The first two tensors correspond to a vector field pointing to the object's center, whereas the third tensor estimates the depth. They make use of a hough voting layer to find the object's center. Here, each vector in this vector field votes for a line of image pixels, and the pixel with the most intersections is the center. If there are multiple instances of the same object, they select all locations over a certain threshold. Another advantage to direct regression is that the center does not have to be visible since all other pixels vote for the center. All pixels that voted for the center are considered inliers and are used to make the final depth prediction. The inliers also define the RoIs, which they use to make the rotation prediction[1]

2.8.3 Evaluation metric

Yu Xing et al.[1] evaluate PoseCNN with the average distance metric proposed by Hinterstoisser et al., which computes symmetric and non-symmetric objects differently.

$$ADD = \frac{1}{m} \sum_{x \in M} ||(R'x + T) - (Rx + T)||^2 \tag{2.5}$$

$$ADD - S = \frac{1}{m} \sum_{x_1 \in M} min_{x_2 \in M} ||(R'x_1 + T) - (Rx_2 + T)||^2 \tag{2.6}$$

M again denote the vertices and m the number of vertices on the mesh. In addition to the rotation, we also take the translation T into account.
The prediction (R, T) is correct if the error is smaller than a certain threshold which is 10% of the object diameter in the case of the YCB-[1] and OccludedLINEMOD-dataset[25]. Using multiple thresholds and calculating the area under the accuracy-threshold curve more accurately represents how a method performs on incorrect predictions[1]

2.9 Continuity of Rotations Representations

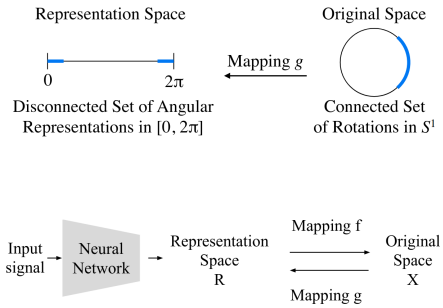


Figure 2.7: Motivation and application of continuous representations. A discontinuous representation is harder to learn because both bounds correspond to the same rotation. To improve this Yi Zhou et al. propose a bijection from the representation to the original space. Image taken from [6]

There are many ways to represent rotations, such as Euler-angles, quaternions, etc.. We say that our network predicts a rotation in the representation space. Since the rotational part of an object's pose is described by 3×3 rotation, we need a mapping f from the representation space to the $SO(3)$ group. However, some representations perform better than others. Yi Zhou et al. argue that this points to discontinuities in the representation space because smoother functions are easier to approximate. They introduce a new definition for continuous representations and show that representations in 4 or fewer dimensions are discontinuous.

Definition 1 *Let R be a subset of a real vector space equipped with the Euclidean topology called representation space. Let X be a compact topological space called original space. Define the mapping to the original space f: $R \rightarrow X$, and the mapping to the representation space g: $X \rightarrow R$. We say (f, g) is a representation if for every x X, f(g(x)) = x, that is, f is a left inverse of g. We say the representation is continuous if g is continuous*

The representation (f, g) defines a homeomorphism between R and X, a continuous bijection with a continuous inverse. One can imagine this as bending and stretching one space into the other. In the context of neural networks we prefer g to be continuous, to create a continuous training signal making it easier for the network to learn.
To show an example of a discontinuous representation consider $SO(2)$. Any 2D rotation can be expressed by

$$M(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{2.7}$$

with θ being the angle we rotate with. Let $R = [0, 2\pi]$, we can define the mapping $f : \theta \mapsto SO(2); \theta \mapsto M(\theta)$. But then the mapping $g : SO(2) \mapsto \theta$ is discontinuous, because the zero rotation, defined by the identity matrix, has to map to 0 and 2π respectively, which is visualized in Figure 2.7. To make this representation continuous M needs to be defined as $M(\theta) = \begin{bmatrix} cos(\theta) & sin(\theta) \end{bmatrix}$ [6].

2.9.1 Continuous Representation

With the definition above, Zhou et al.[6] proved that there are no continuous representations in 4 or fewer dimensions. They develop a method to create continuous rotation representations using the Gram-Schmidt orthogonalization process in the representation space. This way they are able to represent $SO(n)$ in $n^2 - n$ dimensions. Consider $M \in SO(n)$ with

$$M = \begin{bmatrix} | & & | \\ a_1 & ... & a_n \\ | & & | \end{bmatrix} \tag{2.8}$$

g_{GS} drops the last column and f_{GS} uses Gram-Schmidt to compute the last column.

$$g_{GS}(M) = \begin{bmatrix} | & & | \\ a_1 & ... & a_{n-1} \\ | & & | \end{bmatrix} \tag{2.9}$$

$$f_{GS}\left(\begin{bmatrix} | & & | \\ a_1 & ... & a_{n-1} \\ | & & | \end{bmatrix}\right) = \begin{bmatrix} | & & | \\ b_1 & ... & b_n \\ | & & | \end{bmatrix} \tag{2.10}$$

$$b_i = \begin{bmatrix} \begin{cases} N(a_1) & if\ i = 1 \\ N(a_i - \sum_{j=1}^{i-1}(b_j \cdot a_i)b_j) & if\ 2 \leq i < n \\ det \begin{bmatrix} | & & | & e_i \\ b_1 & ... & b_{n-1} & \vdots \\ | & & | & e_n \end{bmatrix} & if\ i = n \end{cases} \end{bmatrix} \tag{2.11}$$

$N(\cdot)$ denotes a normalization function and $e_1, ..., e_2$ are the n canonical basis vectors. Since we use $SO(3)$ this reduces to a simple cross-product $b_1 \times b_2$. Zhou notes that using their 6D representation is useful because the resulting matrix is

orthogonal, whereas applying orthogonalization as a post-process prevents applications like forward kinematics. Empirical results in 3D point cloud estimation and inverse kinematics show that it performs better than other representations and converges quicker [6].

2.9.2 Singular Value Decomposition for Deep Rotation Estimation

	Mean (°)	Med	Std
3D-RCNN	35.50	13.21	46.55
M_G	31.60	16.70	41.86
Euler	41.35	27.44	37.73
Axis-Angle	32.30	19.74	34.70
Quaternion	26.92	14.39	32.92
S^2-Reg	27.36	15.41	33.17
5D	25.18	13.40	32.10
6D	22.60	11.51	31.24
SVD-Inf	21.38	11.41	**29.35**
SVD-Train	**21.25**	**11.14**	30.28

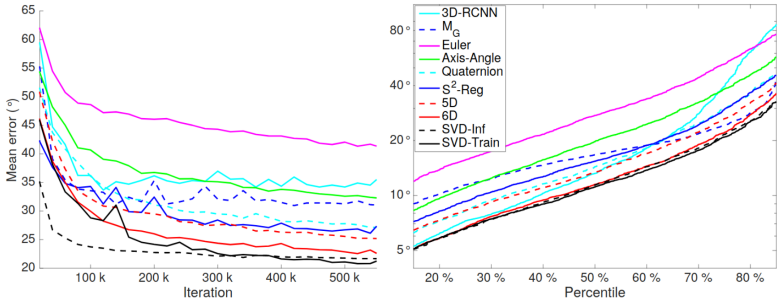


Figure 2.8: Comaprison of different rotation representations for the task of 3D pose estimation from 2D images. Image was taken from[26]

Instead of predicting a 6D output, Levinson et al.[26] predict a 9D output and use a singluar value decomposotion (SVD) to map onto $SO(3)$. The 9D output represents the entries of a 3×3 matrix M with the singular value decomposition $U\Sigma V^T$. To perform special orthogonalization, they compute

$$SVDO^+(M) = U\Sigma' V^T, \; where \, \Sigma' = diag(1, ..., 1, det(UV^T)) \tag{2.12}$$

SVDO$^+$ is smooth and differentiable, except for the case $det(M) = 0$ or $det(M) < 0$ and its smallest singular value has multiplicity greater than 1 and therefore not a continuous representation. Levinson et al. provide a variant SVD-Inf applying SVDO$^+$ only during inference and the training loss directly to M and argue that this is a continuous representation. In Figure 2.8 they show that their 9D representation outperforms quaternions and the 6D representation in tasks like pose estimation from point clouds or 2D images [26].

2.10 Feature Learning on Point Sets

Knowing the 3D geometry provides essential information to our neural networks. Meshes or point clouds represent 3D information, but extending 2D learning approaches to 3D is challenging. Recent advances use Graph Neural Networks to

exploit the connection between vertices and defining pseudo-coordinates for a convolution-like operation. Another approach is learning features from a point cloud. Qi et al. introduce the PointNet++ architecture, which captures a point cloud's global and local structure and encodes it in a feature vector. The main problem in processing point clouds is the permutation invariance. The geometry stays the same even if we scramble all points in our point cloud $\{x_1, ..., x_n\}$. Since the standard 2D convolution exploits the information in neighboring pixels, one can see that the permutation invariance is a challenge. One way is to turn the point cloud into a voxel grid, creating an underlying euclidean structure that makes applying a convolution possible. On the one hand, big voxels create artifacts and loose information. Small voxels, on the other hand, render the data unnecessarily voluminous. Charles Qi et al. introduce the idea of aggregating the features learned on the point set using a symmetric function making the resulting architecture permutation invariant.

$$f(x_1, ..., x_n) \approx g(h(x_1), ..., h(x_n)) \tag{2.13}$$

With $f : 2^{\mathbb{R}^{\mathbb{N}}} \mapsto, h : \mathbb{R}^N \mapsto \mathbb{R}^K$ and $g : \mathbb{R}^K \times ... \times \mathbb{R}^K \mapsto \mathbb{R}$ being a symmetric function. For their symmetric function g, they use a max-pooling function, and a multi-layer perceptron approximates h. This approach already showed good results; however, Qi et al. take inspiration from CNNs where local features are grouped and processed into higher-level features. Small neighborhoods are processed to produce a new set with fewer elements. They again sample a neighborhood from the extracted points and get fewer points which the network considers more important with each step. This way, they capture local and global features[20, 27, 22].

3 Related Work

3.1 6D Pose Estimation

Detecting 3D objects and estimating their pose has many real-world applications like robotics and AR, underlining the importance of this research area. The task is finding the 3D orientation and position in the camera frame given an RGB or RGB-D image, which is an inherently ill-posed problem because we try to reason about 3D objects with only 2D (2.5D) information available. We further assume that we have access to the 3D CAD models and the exact object sizes. With depth cameras like Mircosoft Kinect being so cheap, a common practice is to refine the poses with the Iterative Closest Point algorithm (ICP)[1], but one can also refine the pose using only the RGB image with render-and-compare methods[28]. Many papers focus on the pose estimation of object instances[1, 29]. However, recent papers try to take it one step further and generalize to a whole category of instances.

3.1.1 Instance-Level 6D Pose Estimation

Instance-Level pose estimation focuses on estimating the pose of objects available during training. A large body of work[2, 1, 25, 30] focuses on hand-scale items like drillers, mugs, and bottles, but there has also been research on bigger objects like furniture or cars[31, 32]. Pose estimation approaches can be classified into template matching and regression-based methods.[2, 1]

Sparse Feature-based and Template Matching Methods Traditonally, researchers used template matching methods by scanning the image with a template and computing a distance metric to find the best match[33]. Others use feature-based methods to estimate the 6D pose by finding mesh-to-image correspondeces. The MOPED framework[30] detects keypoints and then uses an iterative refinement process to estimate the pose that fits the image features. Template methods are still used today by matching 3D objects with pointclouds. For example, Yu et al.[24] combine templates with deep learning by predicting a segmented point cloud from multiple images with a deep neural network and then align 3D object

models to this point cloud. These methods achieve impressive accuracies for clearly visible objects, but suffer if the objects are partly occluded, which is common in warehouse settings[25, 30, 24, 34].

Regression-based methods PoseCNN or EfficientPose use a deep neural network to directly predict a 3D rotation and a 3D translation from image pixels. To predict the rotation, these methods first predict a RoI by segmenting the image or predicting a 2D bounding box. They then pool the extracted features from the region and regress them to the rotation and translation. Since they do this regression step sequentially for each RoI, Capellen et al.[35] propose to parallelize this with a pixel-wise rotation estimation. For each pixel they predict a quaternion and then average over the prediction. Others predict the pixel to object surface correspondeces[36] and use the RANSAC algorithm to estimate the pose. Deep learning approaches work well with varying lighting conditions and can also handle occlusions better. However, they struggle with symmetric objects because of shape ambiguities[1, 8, 37]. PoseCNN solves this problem by introducing a new loss function that can handle symmetric objects. Many approaches also use a refinement process like ICP when depth is available [1, 29, 2].

3.1.2 Category-Level 6D Pose Estimation

The problem with instance-level pose estimation is that they cannot be used in general settings because many objects have not been seen during training. For category-level pose estimation, we train on known instances of a category and expect to transfer this knowledge to unknown instances. There has been some progress on big room-scale objects but with some constraints. The rotation predictions are only along the gravity direction, and they do not deal with smaller objects which are more present in robot manipulation task like the Amazon Picking Challenge[24]. Wang et al.[2] extend this to smaller objects. They define a Normalized Object Coordinate Space (NOCS), which can be imagined as a representative for each category. They then train their network on the perspective projection of the NOCS and regress the NOCS map during test time. Together with the depth map, they recover the 6D pose by using a pose fitting method [2, 38, 39, 31].

3.2 Semantic Segmentation

Semantic segmentation assigns labels to image pixels and is a necessary component for image understanding.

Neural Networks for Segmentation Deep learning dominates the field because neural networks can learn features themself, removing the need for tediously handcrafted features. Semantic segmentation can be seen as a dense classification problem, meaning that instead of predicting a single layer for the segmentation, we predict class confidences for each pixel. Notable architectures include U-Net[40], Mask R-CNN[17], and RefineNet. Mask R-CNN builds on top of the Faster R-CNN architecture, which detects objects in an image. They use RoIAlign[17] method to extract the detected objects and then add a few layers to predict the segmentation. Another segmentation network is the U-Net. It first downsamples the image to a feature vector and then upsamples it to a segmentation in the second step while integrating information from the previous stages. The idea behind these skip connections is that we use the feature mapping from image to feature vector to map the feature vector back to an image. Another common design choice is to reduce the number of features when downsampling to create a bottleneck where only essential features get through to the decoder network.
The repeated downsampling trough operations like max-pooling and convolution with stride greater than one reduce the image size by a factor of up to 32, thereby losing the finer image structure.[23]
Noh et al. try to solve this problem by learning a deconvolution that upsamples the feature map to a higher resolution.[41] Line et al. note that the deconvolution cannot recover features from previous stages because they are lost during the downsampling process. Other approaches [42] capitalize on these mid-level features to generate higher resolution feature maps with the idea that the middle layers describe object parts while retaining spatial information. However, Line et al. argue that all feature levels are helpful for higher resolution segmentation [23] Their RefineNet uses a combination of short and long-range residual connections to exploit features from multiple scales of the same image. This way, they can refine a coarse-high level segmentation to a higher resolution.[17, 23, 40]

3.3 Canonical Representations

Pose estimators and 3D deep representation learning methods require large amounts of annotated data that include meshes, point clouds, or implicits. ShapeNet and other datasets that contain 3D models tend to have an underlying bias. These datasets canonicalize their models to a unit bounding box with their center at the origin and objects semantics adjusted to the coordinate system's axes. For example, the x-axis for bicycles always points to the front wheel and the seat's z-axis. The network's prediction quality suffers if the input object is not canonicalized. He

Wang et al.[2] try to solve this problem by learning a mapping from image pixels to the NOCS of the corresponding category. This approach still requires creating a canonical model and the NOCS maps and is therefore trained supervised. Sun et al.[43] introduce canonical capsules to learn a canonical frame unsupervised. They train their architecture on pairs of randomly rotated 3D point clouds to generate a K-part decomposition into K keypoints. After that, they train a deep network to output a transformation to their canonical coordinate system by regressing the descriptors to each other. We also learn a canonical frame unsupervised, but we minimize the point clouds' shape difference instead of creating a decomposition and matching the descriptors. Our module is simple to implement in a pipeline, requires no supervision, and can optimize the canonical frame to the present task [43, 2].

3.4 Training Data Generation

To train a neural network, we need large amounts of annotated data. There are many real-world datasets for object segmentation or detection, with COCO[44] and ImageNet[45] being two examples.

Real World Data Sets for Pose Estimation For other tasks like 6D pose estimation, there are only a few real-world data sets. One often referenced is the LINEMOD[25] dataset, which contains cluttered scenes with large occlusions and poorly textured objects. This dataset, which has 1,000 images, is still small compared to COCO's 80,000 training images. The lack of large datasets is due to the cost and time it takes to annotate data with 6D poses. To annotate an image, a human has to align a mesh with the object in the presented image. This work is tedious and prone to human error, especially when dealing with occlusions. To deal with this absence of training data, PoseCNN introduced their YCB-dataset with 21 objects and 133,827 images with a few examples shown in Figure 3.1. However, it is not useable for tasks like object-category pose estimation because the data set only contains one instance per class [44, 45, 1, 25].

Synthetic Data To alleviate the issues in realworld datasets, synthetic data sets are used. Synthetic data does not take much time to create and is perfectly annotated even for occluded objects. Many of these synthetic data sets do not model sensor noise, material, and lightning, creating a gap between real-world and synthetic data. Different approaches are trying to bridge this gap, like rendering objects over real background images. The main problem here is that some objects

Figure 3.1: Image samples from the YCB-Dataset[1]. The data set consists of 92 unique scenes.

do not have any context to their environment and look like they are flying. Tobin et al.[46] argue that it is possible to overcome this gap by randomizing rendering in simulations. Enough variations of object poses, object shapes and lightning, will create scenes that are close to the real world. They train an object detector solely on multiple low-fidelity rendered scenes with non-realistic textures and can achieve an accuracy of 1.5cm on real-world data. We adopt this approach and render scenes with different object positions, occlusions, material appearances, and image noise, creating physically realistic scenes that always look different [46, 2, 1].

CAMERA He Wang et al.[2] solve the object context problem by first detecting a surface in the image and then placing rendered objects on top of it. Figure 3.2 shows some qualitative examples of the generated images. The data set consists of 31 real scenes with six object categories from ShapeNetCore[47] and one distractor category. They also add 4300 images of real-world annotated data to improve the quality of their algorithm further. They provide the RGB-D image with the corresponding NOCS map. They decided not to include the 6D pose directly but provide code to extract it from the NOCS map. The resulting ground truth does not fit the actual pose in the image as shown by [48].

Figure 3.2: Examples from CAMERA[2]. He Wang et al. render the objects in blender and place them on surfaces in real images.

DeepCPD Diego Rodriguez et al.[4] introduce the DeepCPD mesh data set to learn 3D non-rigid registration. They want to learn intra-class deformations and divide this data set into four object categories sprayer, bottle, camera, and drill, which makes it also viable for category pose estimation. They took the models from sketchfab, an online mesh database, and from [2].

4 Our Approach

This section gives insight into how we create our data and explains the network architecture using ideas from the papers mentioned in the previous section.

4.1 Synthetic Dataset

To the best of our knowledge, the CAMERA data set[2] is the only data set for category-pose estimation with hand-scale objects. Other data sets like YCB[1] and LineMod[25] are not suited because they are limited to one instance per category. CAMERA contains both real and synthetic data. Although the CAMERA data set has some nice properties like no flying objects or unusual poses, there are two major disadvantages. First, lack of occlusions. The scenes present in the data set are simple tabletop scenes with almost no occlusion. In real world bin picking scenes, the objects lie in a pile and can use each other as support to get in uncommon pose configurations. Not training on these kind of data results in a underspecification of our model to real world scenarios [49]. In tasks like the Amazon Picking Challenge[24], the network has to deal with high occlusions, which makes it essential to include them in our training. Secondly, lack of shadows and unrealistic lighting. The synthetic objects are rendered in blender using random lightning sources to simulate indoor settings. However, the random light sources do not fit the conditions in the image, which is especially noticeable with objects in shaded areas. Also, the lack of object shadows is a common theme in many images.
We overcome these limitations by creating a photorealistic synthetic dataset using the meshes provided by DeepCPD[4] using Stillleben[3].

4.1.1 Realistic Cluttered Scenes using Stillleben and DeepCPD

We use the four categories driller, bottle, camera, and sprayer from [4] to create 70 object combinations for training, 10 for validation, and 15 for testing. We exclude four meshes from each category to create our evaluation and test set with two meshes and use the remaining meshes for training. Each scene contains one object per category and one object from the YCB-dataset to increase the robustness of our

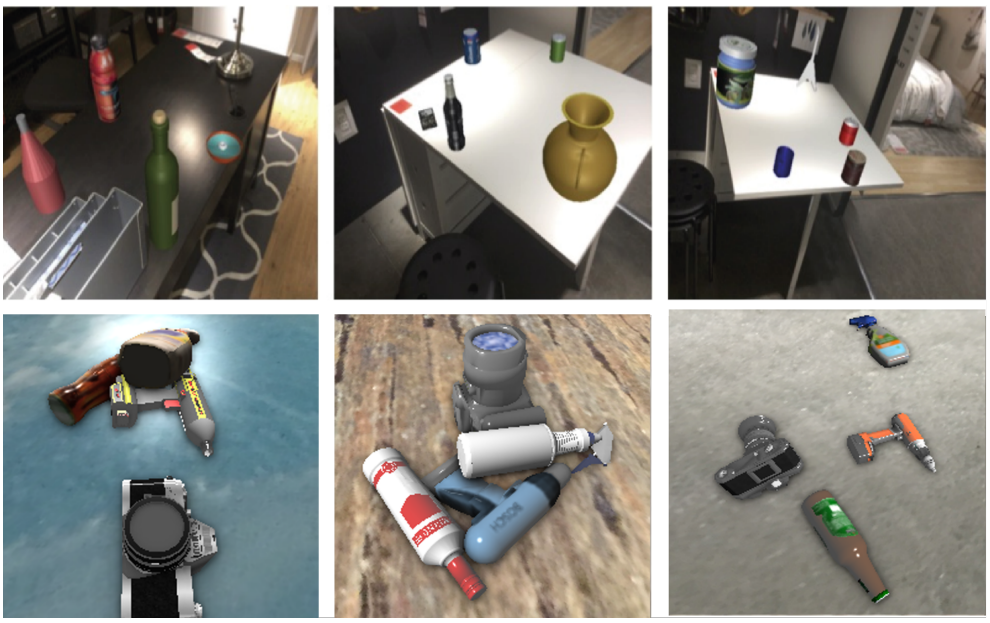


Figure 4.1: Comparsion of our images to CAMERA. The top row shows three samples taken from CAMERA[2]. Objects are realistically placed but lack shadows, correct lightning and occlusions. The bottom row shows three examples of our data. Our cluttered scenes introduce occlusions and have correct lightning conditions.

network. The objects are then placed in the world by the Stillleben arrangement engine. We choose random values for the ambient light, the lightning position, and a random texture from the TUM texture database[50] and [51]. Stillleben then renders all images with the same intrinsic in a resolution of 256×256. At training time, we apply noise, chromatic aberration, blur, and random exposure.
Figure 4.1 shows that the object interactions are more complex compared to CAMERA, with multiple objects overlapping, creating poses that do not occur in CAMERA. These challenging conditions are also present in warehouse tasks like the Amazon Picking Challenge [24].

Z-Translation and Origin Heatmap

Our network learns to predict the rotation and segmentation, a heatmap for the object's origin, and a translation in the Z direction. Stillleben provides the ground truth segmentation in addition to the rendered image. Inspired by PoseCNN, we use the fact that we can infer the translation in the X and Y direction if we know the location of the object's origin in the image and the Z-translation. We encode the object's distance by assigning its Z-translation to its segmentation label and project the origin in the camera frame onto the image plane. We then place a 2D

gaussian bell with $\sigma = 0.001$ over this pixel (c_x, c_y) with

$$f(x, y) = e^{-\frac{(x-c_x)}{2\sigma^2}+\frac{(y-c_x)}{2\sigma^2}} \tag{4.1}$$

4.1.2 Limitations of Synthetic Data Generation

We can generate physically realistic data and train our network using this image generation pipeline. However, this approach can be very time-consuming depending on the quality and amount of the object meshes. We recommend creating a base dataset of saved images, which allows controlled comparison of different models, and create each batch with a combination of saved and online generated data for better performance and overfitting prevention. It is important to mention that synthetic data only mimics real data. For example, camera noise and lightning conditions are more complex in real-world data. A CNN trained on real data will therefore perform better in real-world tasks. However, recent works like Domain Randomization by Tobin et al.[46] show that it is possible to get similar results on synthetic data by randomizing the parameters, e.g., noise, lighting conditions, and object appearance. Some of the scenes are close to reality and are used to train the network. This way, it is possible to bridge the reality gap and get similar performance [3, 46, 52].

4.1.3 Canonical Coordinate System

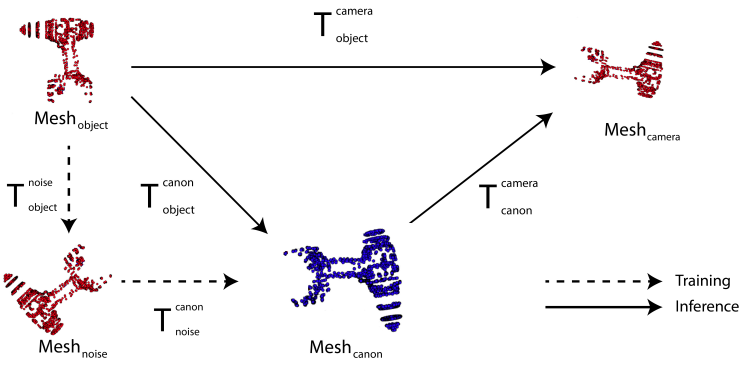


Figure 4.2: Motivation of the canonical coordinate system. Instead of predicting the transformation from object to scene directly, we predict an intermediate coordinate system where all category instances are aligned. During training we have to take the noise transformation into account.

While creating a mesh, the mesh creator defines an arbitrary coordinate system

that defines the position of mesh vertices. The axes' orientation and the location of this system can vary drastically between data sets, which results in our network learning a single transformation for each object instance if we want to perform 6D pose estimation. Suppose we find a common coordinate system for each category. In that case, instead of defining the canonical coordinate system manually, we only have to learn a transformation from the category to the camera coordinate system. Creating a common representation can be done by aligning the models by hand or by creating a normalized representative of a category[2]. However, instead of creating a canonical frame, which might not be optimal for the specific object category, we can also learn a canonical coordinate system.

We assume that we have meshes available and then learn a transformation to this canonical coordinate system. We split the task of 6D object pose estimation into two individual transformations: transformation from object to canonical coordinate system and canonical coordinate system to camera coordinate system as shown in Figure 4.2.

$$T_{obj}^{camera} = T_{canon}^{camera} T_{obj}^{canon} \tag{4.2}$$

4.1.4 Optimizing the Canonical Coordinate System

Figure 4.3: Meshes from the DeepCPD data set. Objects of the same category are similiar in their geometry which allows us to introduce a loss that measures the geometric differences. Image was taken from [4]

We design our network to have as much freedom as possible when learning a canonical representation. However, many objects from different datasets are not

aligned. Some of the commonly used convetions for defining the origin of the meshes are geometric mean, center of the 3D bounding box, center point on the base of the object, etc. These differences do not make the task of finding a common coordinate system obvious for the network. Therefore, we define an auxiliary loss to guide the network in learnig the canonical coordinate system. The auxiliary loss is minimized along with the 6D pose estimation loss. Objects of the same category are similar in shape with a few examples shown in Figure 4.3. We exploit these geometric features by minimizing the ShapeMatchloss between all instances of the same category. We do not explicitly specify how the objects should be oriented or where the origin is located, instead we let the network learn its own canonical coordinate sytem self-supervised.
Formally, let $C = \{c_1, c_2, .., c_n\}$ be the set of all categories, M the set of all meshes and $M_{ci} \in M$ the i-th mesh of category c, $T_{ci} \in T$ the corresponding transformation to the canonical frame.N is the number of meshes in a category.

$$SLoss^m(M_1, M_2, T_1, T_2) = \frac{1}{2m} \sum_{x_1 \in M_1} min_{x_2 \in M_2} ||T_1 x_1 - T_2 x_2||^2 \tag{4.3}$$

$$CanonicalLoss(M, T) = \frac{1}{N} \sum_{c \in C} \sum_{i \in I_c} \sum_{j > i, j \in I_c} SLoss^m(M_{ci}, M_{cj}, T_{ci}, T_{cj}) \tag{4.4}$$

4.1.5 Adjusting Ground Truth Data

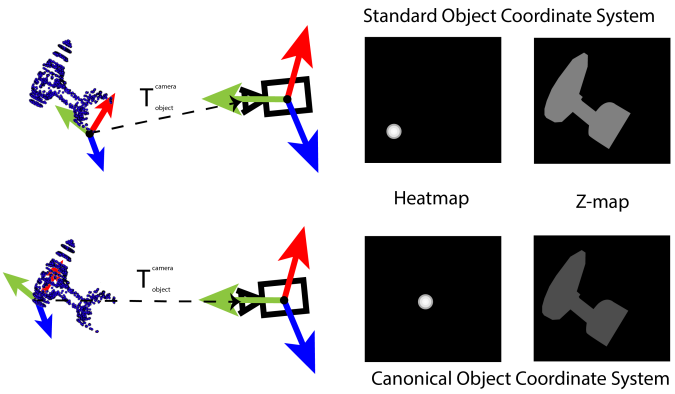


Figure 4.4: Adjusting the ground truth. During the training process, the network actively learns the canonical coordinate system. Thus we create the ground truth after the networks prediction of the canonical frame.

In section Section 4.1.1, we mention that we create a heatmap and z-map to

estimate the translation of our object. If we do not use a canonical coordinate system, we can use the translation of the transformation given by

$$T_{obj}^{camera} = T_{world}^{camera} T_{obj}^{world} \tag{4.5}$$

which we can calculate with Stillleben. However, our canonical representation changes the position of the standard coordinate system, and we therefore need to adjust the ground truth heatmap and z-map. This point is illustrated in Figure 4.4. To solve this problem, we first use the predicted transformation T_{obj}^{canon} and transform it to the original object coordinate system. Using this, we calculate the transformation to the camera frame

$$T_{canon}^{camera} = T_{world}^{camera} T_{obj}^{world} T_{obj}^{canon\,-1} \tag{4.6}$$

and project the new translation on our image plane to the pixel (c_x, c_y). Note that we apply noise transformations T_{obj}^{noise} to our input meshes and therefore need to compute

$$T_{canon}^{camera} = T_{world}^{camera} T_{obj}^{world} (T_{obj}^{noise} T_{noise}^{canon})^{-1} \tag{4.7}$$

4.1.6 Regularization

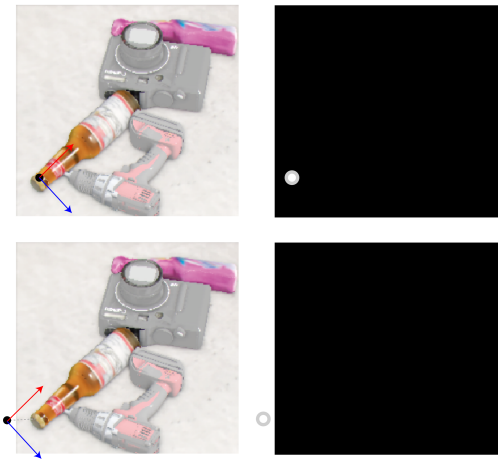


Figure 4.5: An origin that lies beyond the image boundaries cannot be detect with our heatmaps. We therefore need to pull it closer to the object.

Another problem is that the network can place the object's origin far away from the object since we do not specify its position. Figure 4.5 shows the worst case scenario. The origin is not in the image, making it impossible to detect it with a

heatmap. We introduce a regularization term that pulls the origin closer to the center of mass. One way of regularization is penalizing transformations that push the object's center of mass M too far away from the origin. In this case, the regularization term is the squared distance to the origin.

$$L_{reg} = \begin{cases} 0 & \text{if } ||M||^2 < t \\ ||\mathrm{M}||^2 & \text{else} \end{cases} \tag{4.8}$$

Another way is minimizing the squared closest distance from one point p on the mesh M to the origin $(0, 0, 0)$ and set the error to 0 if we are below a certain threshold t.

$$d_{closest} = min_{p \in M}(p^2) \tag{4.9}$$

$$L_{reg} = \begin{cases} 0 & \text{if } d_{closest} < t \\ \mathrm{d}_{closest} & \text{else} \end{cases} \tag{4.10}$$

This way, the origin is placed somewhere near the mesh while allowing the network to move the origin along the mesh surface. The downside is that we cannot move the origin far inside voluminous objects like a ball or a box.

4.2 Network Architecture

In this section we explain the network architecture, which can be seen in Figure 4.6, that we use to learn a category-level canonical coordinate system and perform object-category 6D pose estimation.

4.2.1 Canonical Rotation Estimator + Translation - CaRET

Our CaRET architecture utilizes two stages that perform different tasks. The first stage predicts the objects distance, a heatmap, and a segmentation. Each category has its own rotational branch to extract relevant features from the features of the first stage.

Features Extraction and Translation Estimation We learn two transformations T_{obj}^{canon} and T_{canon}^{camera}. The first one transforms our mesh to our canonical coordinate system and the second one estimates the transformation from our canonical coordinate system to the camera frame. Yu Xiang et al.[1] decoupled the pose

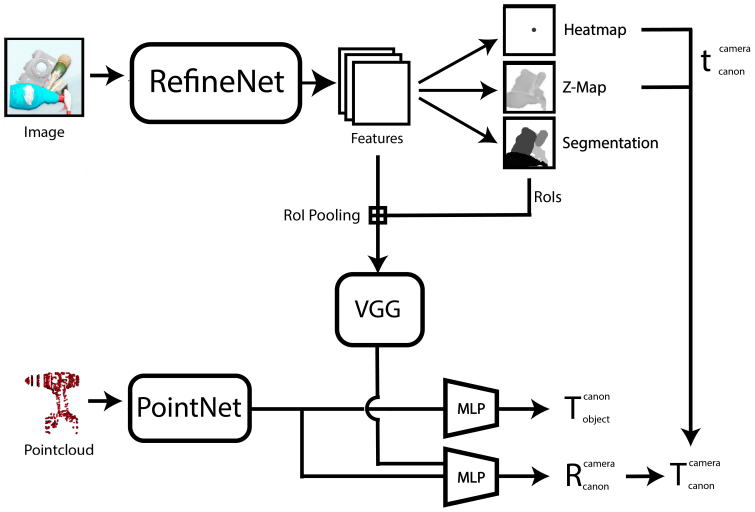


Figure 4.6: Pipeline overview. Our pipeline uses a RefineNet architecture to predict a heatmap, z-map, and segmentation. Heatmap and z-map estimate the translation from the canonical frame to the scene. A slightly altered VGG extracts process the features from the backbone. One MLP predicts the transformation to the canonical frame while the other estimates the final rotation. Both use the PointNet features, but only the second MLP uses image features.

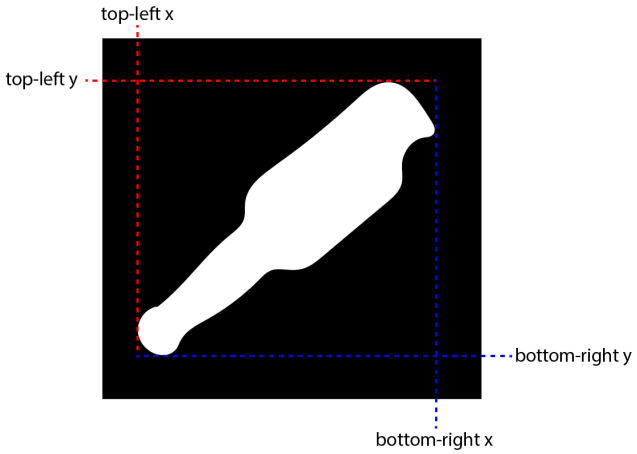


Figure 4.7: Computing the RoI from a segmentation. We find the top-left and bottom-right corner of our segmentation to extract the bounding box.

estimation into rotation and translation estimation and we follow this approach. The first stage of our network consists of a RefineNet backbone, pretrained on ImageNet. Three subbranches then predict the segmentation, a heatmap for the origin, and translation in Z direction using the RefineNet features. The segmentation branch's output is an (N+1)×128×128 feature map where N corresponds to the number of classes and the background. Figure 4.7 visualizes the RoI extraction process. We divide each channel by its maximum and create a segmentation mask for each object. We refine noisy prediction, using a 3×3 erosion, followed by a dilation of the same size to get more accurate regions of interest. Then we calculate the object's bounding box by finding the top left and bottom right corner of the segmentation.
Let $seg(x, y)$ be the value of our segmentation at pixel (x, y), the set of object pixels is defined by:

$$S_x = \{x \mid \exists\, y : seg(x, y) = 1\} \tag{4.11}$$

$$S_y = \{y \mid \exists\, x : seg(x, y) = 1\} \tag{4.12}$$

Then the bounding box is defined by its top left and bottom right corner

$$tl = (min(S_x), min(S_y)) \tag{4.13}$$

$$br = (max(S_x), max(S_y) \tag{4.14}$$

We use our heatmaps and Z-translation maps to estimate the translation of our object. The Z-translation map is of size N×64×64 and predicts the translation in the Z direction for each visible object category pixel. The heatmap is of size N×32×32 and predicts a probability for each pixel to be the origin. To get the translation of our objects, we need to decode both predictions. We first calculate the object's distance by taking the mean over the object pixels. Secondly, we resize the heatmap to the image resolution and compute the $argmax$ giving us the pixel where our network expects an origin. Reprojecting this origin into 3D space gives the translation $t_{canon}^{camera} = [t_x, t_y, t_z]$ for each object.

Canonical Frame & Rotation Estimation Before we use the RoIs, we predict T_{object}^{canon} in our canonical branch. We use the extracted shape signature from a pre-trained PointNet++ network, which encodes information about our mesh's geometry. We add a small MLP that takes in the signature and outputs $T_{object}^{canonical}$ in our chosen rotation representation plus three parameters for the translation.
The segmentation does not contribute to the translation. However, it is necessary to extract the regions of interest from our RefineNet features using an RoI pooling

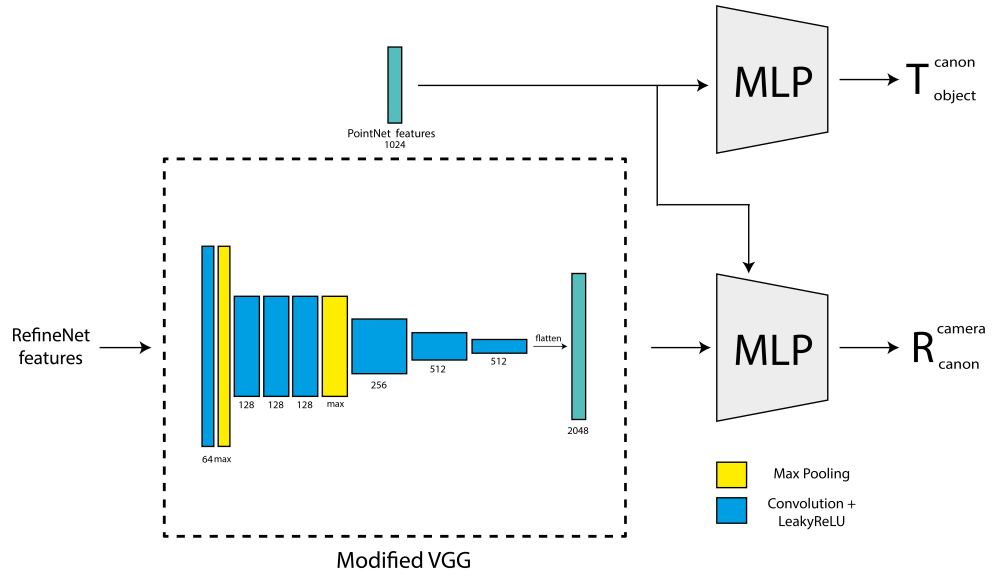


Figure 4.8: Category specific rotation branch. The modified VGG learns category specific features which are flattened and passed to an MLP. The MLP processes the image feature vector first and then adds the PointNet features. The output is a rotation in our choosen rotation representation.

layer introduced in Mask R-CNN. We resize the RoI to 128×128, resulting in a 128×28×128 feature vector for each category. After that, we apply a series of convolutions and downsample the features using a combination of max pooling and 3×3 convolutional layers with stride 2 to create a bottleneck where only essential features get to the next stage. A detailed version of the CNN is illustrated in Figure 4.8. Then, we predict R_{canon}^{camera} the rotation from our canonical frame to our camera coordinate system. We use the extracted image features, flatten them and add them to our predicted PointNet signature and use another MLP to predict the rotation. We then map both rotation representations back into $SO(3)$ and create the transformations

$$T_{canon}^{camera} = \begin{bmatrix} R_{canon}^{camera} & t_{canon}^{camera} \\ 0 & 1 \end{bmatrix} \tag{4.15}$$

$$T_{obj}^{canon} = \begin{bmatrix} R_{obj}^{canon} & t_{obj}^{canon} \\ 0 & 1 \end{bmatrix} \tag{4.16}$$

4.3 Training Implementation

Pose estimation is a complex task that requires many intermediate results. Our network has to predict a segmentation to detect objects and estimate the transformation to the camera and canonical frame. This section gives further inside into the training implementation.

Optimizer & Scheduler We train our architectures with Adam[14], a learning rate of $1e-4$ and the CosineAnnealingLR scheduler[12], which adapts the learning rate after one epoch, which corresponds to 1000 randomly sampled images.

Data Augmentation We use the images generated by our data generator and process them with random noise parameters that influence the blur, exposure, chromatic aberration, and color jitter. We apply noise transformations at each training step that rotate our point clouds with an angle between 0° and 15°.

Loss Functions To optimize our predictions, we apply a $LogSoftmax$ to our segmentation prediction to get values between 0 and 1 and use negative log-likelihood as the loss for for our segmentation branch resulting in the loss L_{seg}. For the heatmap loss L_{heat}, we use a variant of focal loss, which Law et al.[10] introduced in CornerNet. We apply a sigmoid function to the heatmap and clamp the values to the interval $(1e-4, 1-1e-4)$ to prevent NaN values. For the distance error L_z, we calculate the MeanSquared-Error between prediction and ground truth, resulting in the combined loss

$$L_1 = L_{seg} + L_z + L_{heat} \tag{4.17}$$

for our first stage.
Secondly, our second stage calculates $T_{object}^{canonical}$ and $R_{canonical}^{camera}$ in our choosen rotation representation. We convert the representations to rotation matricies and combine $R_{canonical}^{camera}$ and $t_{canonical}^{camera}$ to get T_{canon}^{camera} and optimize this transformation with our $CanonicalLoss$. Because of the translational part of the canonical prediction we include the regularization loss L_{reg} mentioned in section Section 4.1.6 and let it decay. For the final pose in the scene we use the $SceneLoss$ which applies $SLoss^m$ to symmetric and $PLoss$ for transformation to non-symmetric objects. For the $SceneLoss$ we compute the predicted transformed mesh M_{camera} with

$$M_{camera} = T_{canon}^{camera} T_{object}^{canon} M_{object} \tag{4.18}$$

and the ground truth with

$$M'_{camera} = T_{object}^{camera} M_{object} \tag{4.19}$$

Note that because of the applied noise transformation T_{object}^{noise} we actually learn $T_{noise}^{canonical}$. We also weight both losses differently with α for the $SceneLoss$ and β for the $CanonicalLoss$. We set α to 1 for all experiments and varied β between 0 and 1. Our final error for the whole pipeline is

$$L_{final} = L_1 + \alpha L_{scene} + \beta L_{canonical} \tag{4.20}$$

Training Optimization The pipeline is trained end-to-end. However, optimizing the segmentation first and then the rotation stage is more efficient because we use the segmentation for our RoIs resulting in unusable data for the second stage at the beginning of the training. We can divide the second stage into a separately trainable canonical branch and a scene branch. However, backpropagating the scene's loss through the canonical branch might help find a better canonical frame. We prerender the dataset and save it to the disk. This is more efficient than generating images on the fly. Simulating a batch of tabletop scenes and rendering new images takes around 640ms, whereas loading it from the disk takes 10ms. We use the DataParallel interface provided by PyTorch[11] to train our model efficiently using multiple GPUs.

5 Experiments

In this section, we analyze the prediction quality, speed of convergence of different rotation representations, the effect of the weight for the CanonicalLoss during training, and how our pose estimator performs on unseen objects.

5.1 Experiments with CaRE

We use a variant of CaRET that only predicts the rotation component. We abbreviate it with CaRE and investigate the effects of different variables on our model's rotation predictions. The CaRE model directly extracts features from the image instead of taking the RefineNet features as input. To reduce training time and complexity, we limit the task to five instances of the drill category and render each object against a white background with a fixed translation and a random rotation.

5.1.1 Comparison of Rotation Representations

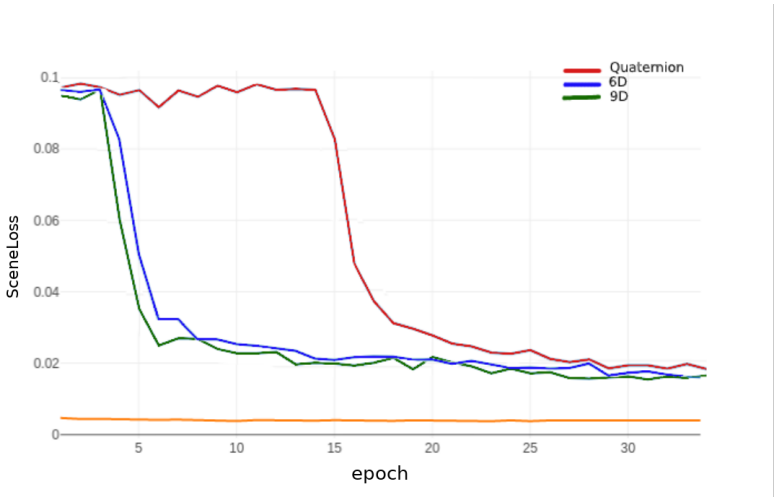


Figure 5.1: Comparison of rotation representations. Higher dimensional rotation representations converge faster.

In section 1.10, we reviewed the 6D and 9D rotation representation. These

representations outperformed quaternions and other rotation representations in various tasks[6, 26]. To investigate the effect of the different rotation representations on our training, we trained the same model with three different rotation representations for 35 epochs and a learning rate of 3e-5. We removed the background plane and other objects and fixed the drill position to simplify and speed up the training.
Figure 5.1 shows that the higher dimensional and continuous representation perform better than quaternions. Quaternions are stagnant for the first 14 epochs and then start to drop, whereas 6D and 9D converge after six epochs. In the beginning, it looks like 9D performs slightly better than 6D. However, if we continue the training, we see that using SVD leads to instabilities in the canonical pose, influencing the final predictions. On the other hand, Gram-Schmidt is stable and outperforms 9D after 225 epochs. Therefore, we chose 6D as our rotation representation in all our further experiments.

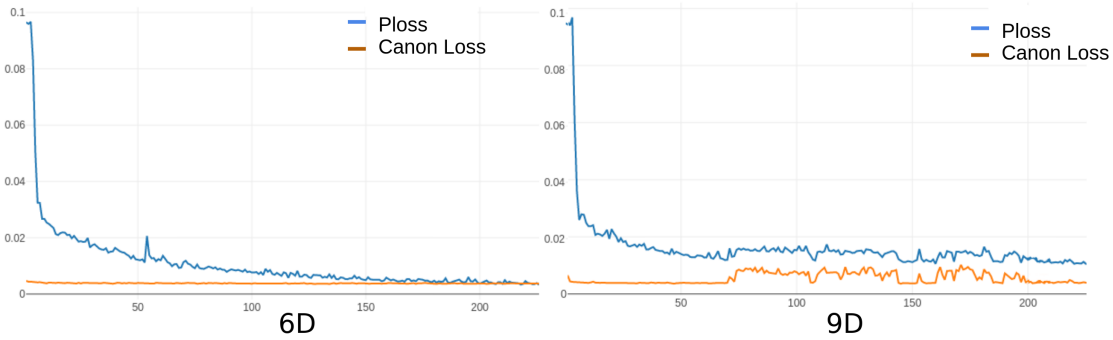


Figure 5.2: Comparison of 6D and 9D rotation representations. The 9D representation becomes unstable leading to fluctuation in the canonical loss which in return influences the final scene prediction. 6D converges smoothly and has an overall better performance

5.1.2 Canonical Loss Weighting

The beta factor weights our CanonicalLoss, and our experiments show that it also impacts the network's ability to generalize to unknown objects. We trained two CaRE models with a beta factor of 0 and 0.5 on drills with a fixed position, random rotation, and no occlusions for 20 epochs. An interesting observation is that our model learns to align the objects without the CanonicaLoss. In figure 5.3 the model_0 learns to align the drill after six epochs, visualized by the drop-off of the CanonicalLoss. The freedom in choosing the position results in a slightly faster convergence for the model_0. One reason for this result is that the meshes

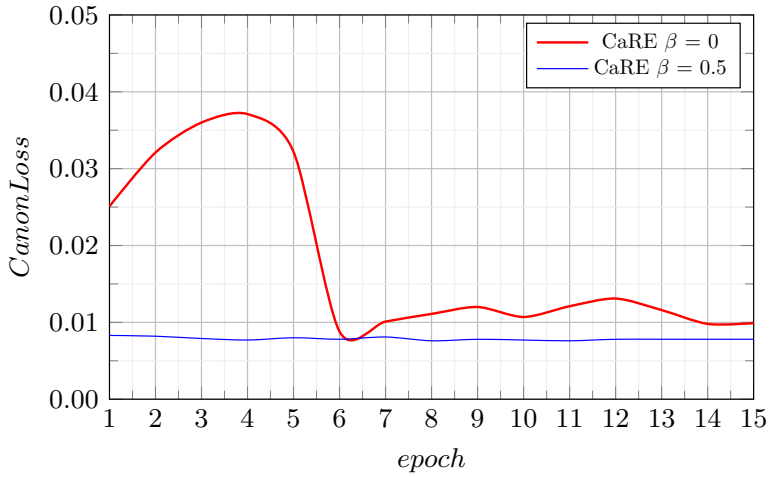


Figure 5.3: Alignment without loss. The model without canonical loss minimizes the CanonicaLoss without us enforcing it.

in our data set are already roughly aligned, and the network only has to learn a minimal transformation. With more significant differences in the initial mesh poses, we would need to pre-align them or use the CanonicalLoss to enforce our definition of a canonical pose.

5.1.3 Evaluation on Unknown Drills

The canonical coordinate system allows the network to learn a rotation to the camera frame for a whole object category. To verify this, we investigate how well our model can generalize to previously unseen instances using the ADD metric. We consider a prediction correct if the ADD error is smaller than 10% of the object diameter. Figure 5.4 shows the rotation evaluation of our model_0 and $\text{model}_{0.5}$ on the drill category. The model that learns the canonical pose by itself achieves an ADD score of 85%, performing better than our other model.

Overfitting with Enforced Canonical Loss Both the β_0 and the $\beta_{0.5}$ model can transform the unknown drills to their canonical coordinate frames, suggesting that the difference lies in the stage where we predict the rotation from the canonical frame to the camera frame. One possible explanation is that the transformation to the canonical and the camera frame requires different or conflicting features from the PointNet. The β_0-model has complete freedom over its canonical coordinate system. Therefore, the PointNet chooses mesh features that help the transformation to the scene, whereas matching points to other instances is a different task. One could argue that these tasks are similar, but the CanonicalLoss requires the

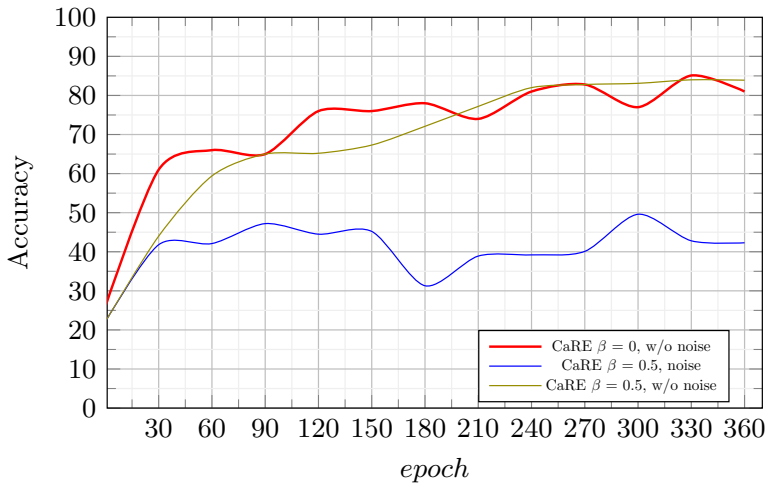


Figure 5.4: Generalization to unknown Instances. The graph plots the number of correct predicition using the ADD metric. The model trained with the CanonicalLoss needs noise to achieve similar performance compared to the model trained without.

alignment to multiple instances instead of one. Both modules want different features with the enforced loss, causing a conflict in the PointNet and overfitting to solve this problem. Introducing rotational noise causes the PointNet to output different features for the same instance, forcing the modules to generalize better.

5.2 Experiments with CaRET

CaRET is the complete pipeline, including the translation estimation. We test this model on our synthetic dataset with four categories, rendered on a surface with a random texture and physically realistic poses.

5.2.1 Evaluation of the Canonical Module

In this section, we analyze the canonical module. We first visualize the learned pose and explain the limits of our module. Then we evaluate the effect on convergence, the metrics on the known objects, and the generalization to unknown instances.

Visualization of the Canonical Pose

To visualize the learned canonical pose, we train our CaRET model with the canonical loss and a rotational noise of 12° on our synthetic DeepCPD data set and visualize the pose after every epoch. In the beginning, we observe that the canonical pose for the test instances of the sprayer category is not the same as the pose for the training meshes. The canonical branch learns to align the objects of the batch because of the CanonicalLoss. After enough iterations, the CanonicalLoss is small enough that the gradients of the SceneLoss have a more significant influence on the canonical pose, therefore, optimizing it to the task of 6D pose estimation. Another noticeable observation is that CaRET tends to move the meshes further away from the origin. This behavior is not desirable because detecting origins outside the object's boundaries is difficult and impossible if the origin is not within the image boundaries, as shown in Figure 4.5.

We introduce a regularization, which decays over time. However, if the regularization weight becomes too small, the network starts to move the meshes further away, and we therefore remove the decay. A representation where the origin is further away might be better for the network, but this is not compatible with our architecture. One could remove the heat- and z-map and directly regress image pixels to a translation vector. However, as the authors of PoseCNN[1] note, this approach cannot detect multiple objects and is not generalizable because objects can appear anywhere in the image.
The next question is, if training without the CanonicalLoss results in a different canonical frame. We train another CaRET model for 500 epochs without the loss and visualize the pose. In Figure 5.6 we can see that the two models learn different coordinate systems, suggesting that there is not one optimal canonical pose and that the losses influence this pose. The $\text{model}_{0.5}$ receives mixed signals from the Scene- and CanonicalLoss. In Figure 5.7 we observe that one drill instance moves away from the others. There is an instance imbalance during training, meaning that one object instance appears more frequently than others. The SceneLoss wants to change the canonical pose and pulls the instance further away from the other objects. The CanonicalLoss, on the other hand, pulls the object back because the other objects still form the majority of the batch. Hence, leading to an unstable canonical pose where meshes are not aligned in the early stages of the training. An improvement could be to only enforce the loss at the beginning of the training or letting it decay over time. However, one could also use the iterative-closest-point algorithm to align the objects once at the start and then learn the canonical pose without the canonical loss. However, the CanonicalLoss benefits the training, which we discuss in the next section.

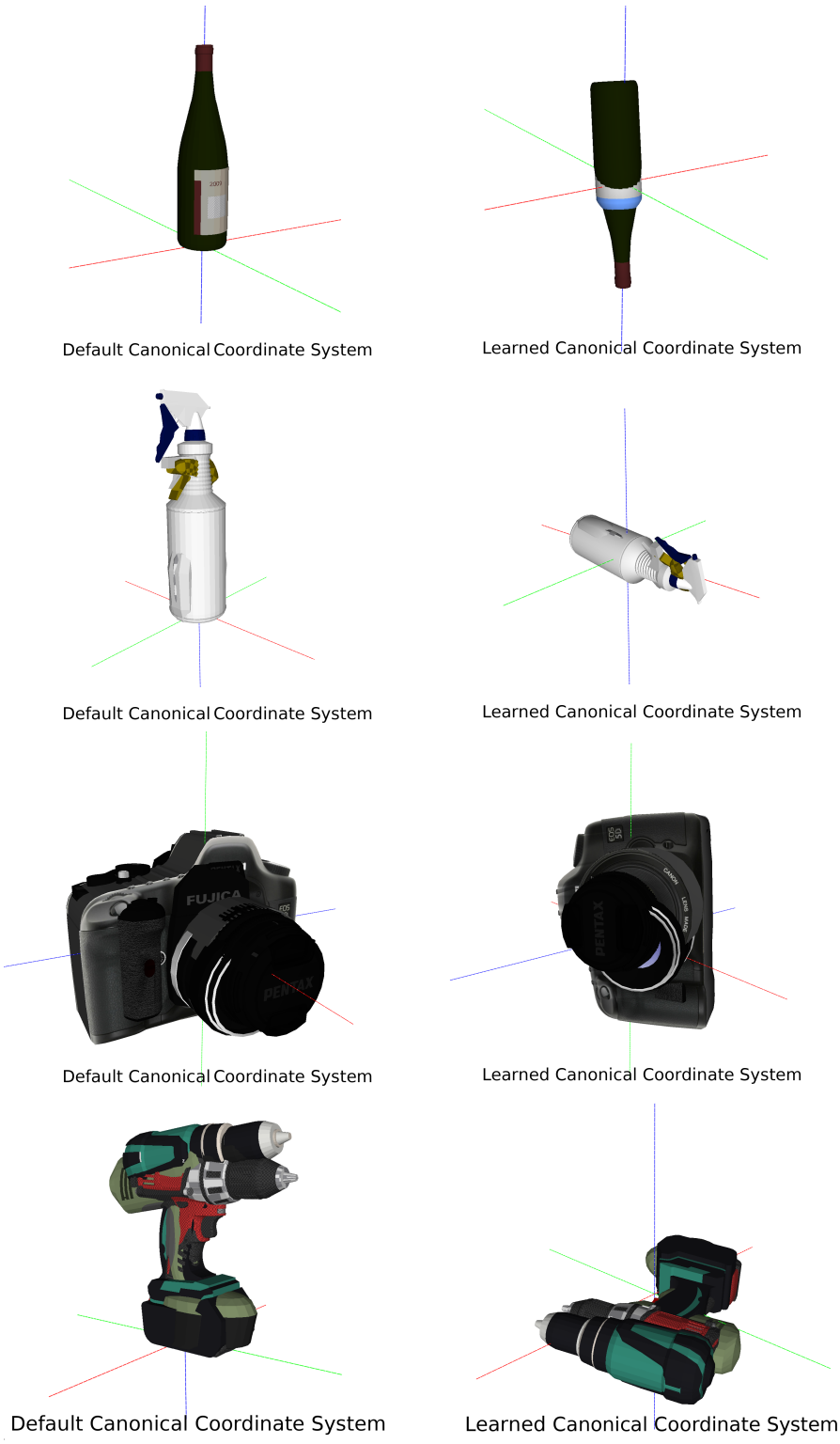


Figure 5.5: Comparison of poses. The left side is the original coordinate system and the right side our learned canonical coordinate system.

Figure 5.6: Comparison of canonical poses. Different betas result in different orientations for the objects.

Figure 5.7: Evolution of the canonical pose during the trainings process. Starting from the inital pose on the left, the SceneLoss pulls the drills apart, but the CanonicalLoss pulls them back together.

Evaluation of Training

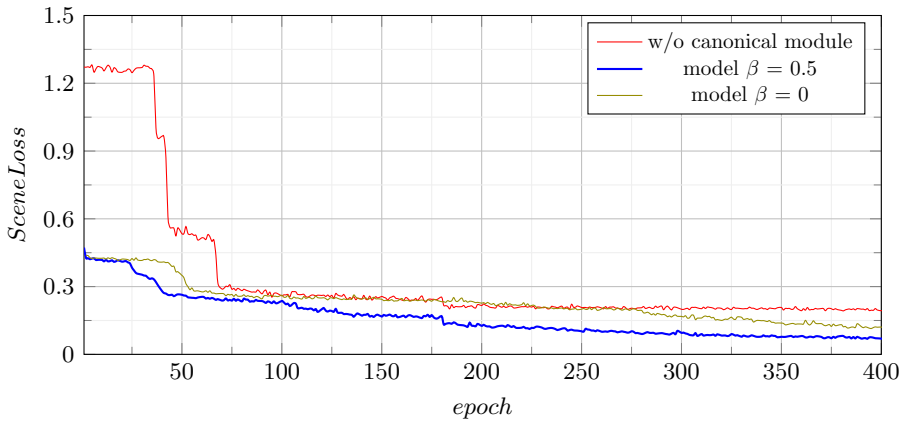


Figure 5.8: Comparision of the trainings loss between the three networks. Adding the canonical module improves the SceneLoss. The canonical loss further improves the model if object instances are not prealigned.

We train three models to analyze the impact of our canonical module on the prediction quality for known instances. The first two models include the canonical module, and we train them with $\beta = 0.5$ and $\beta = 0$. We remove the canonical module for the third model and use this architecture as a baseline. We know that rotational noise helps the module generalize better to unknown instances from training with different betas. We want to investigate the effect of noise on the training and train all models with and without noise. The maximal noise is 12°, and we do not apply any translational noise. The models have to learn the segmentation first, resulting in worse RoIs in the early training stages. In the beginning, the network outputs black segmentations. In this case, we pass the 128×128×128 feature map uncropped to the rotational branch. In Figure 5.8 we can see that the network without the canonical pose starts with a SceneLoss of 1.2 and remains stagnant for 38 epochs before it starts converging. Also, the loss for model_0 is stagnant for 35 epochs. However, the loss is already lower than for $\text{model}_{nocanon}$. When training with the canonical loss, we start converging after 22 epochs, almost twice as early. The higher loss for $\text{model}_{nocanon}$ likely lies within the initialization of the network. Most objects are not standing up but are rotated around the X- and Y-axis, meaning that the network has to learn these rotations first before the loss drops. The $\text{model}_{nocanon}$ can only influence the final transformation with the learned features from the VGG, whereas the canonical pose directly influences the final prediction for model_0 and $\text{model}_{0.5}$ from the start. Therefore, the canonical pose gives the network a better initial starting point for convergence. Figure 5.8 shows that the convergence of the models also differs. The

beta models converge smoothly, whereas the model without the canonical module looks like a step function.

Effects of Rotational Noise

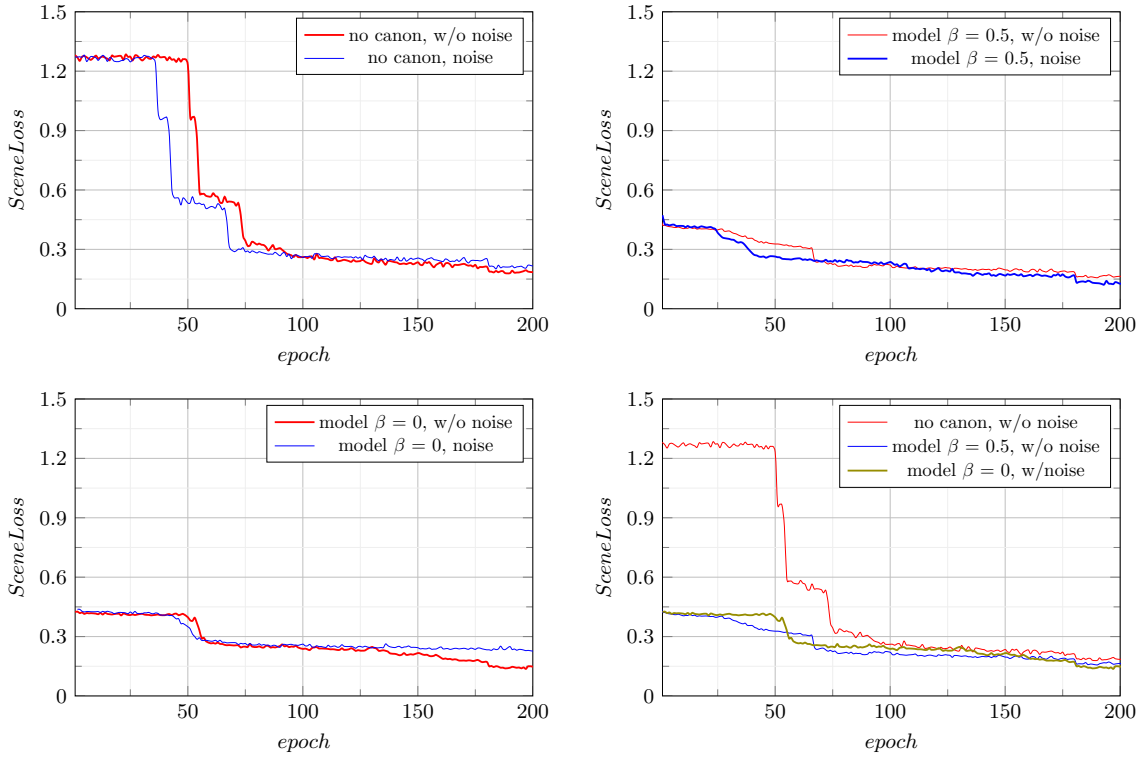


Figure 5.9: Comparison of models with and without noise. Rotational noise in the input makes every model slightly worse except the model that was trained with the canonical loss.

We train the three models without noise to investigate if the rotational noise influences the network's convergence, which we illustrate in Figure 5.9. Noise causes the model without the canonical module to converge earlier but did not improve the training loss. The model trained with β=0.5 is also not significantly influenced by the absence of noise during training even performing better if we apply noise to the input meshes. However, noise affects the β=0 model the most. This difference may be due to the prealignment of object instances in DeepCPD[4]. Without the noise, the model can arrange the objects more freely without having to compensate for the introduced misalignment. Model_0 without noise and $\text{model}_{0.5}$ with noise achieve similar performance on the training dataset and the unknown instances. The loss of the model without the canonical pose is consistently higher

than the other models, as shown in Figure 5.8. This is mainly due to the sprayer category.

Evaluation of Known Instances We create a test set with known instances and evaluate the accuracy of the baseline and the proposed models. We train all models for 1400 epochs and freeze the canonical module after 300 epochs. We use the ADD and ADD-S metric, which takes both translation and rotation into account to compare our models. All models achieve similar results on the drills, but the models with the canonical module achieve slightly better results. Since the bottle category is symmetric, we have to compare the ADD-S scores. Again the models achieve similar scores. We can observe greater differences for the cameras and sprayers, where the models with the canonical modules achieve better scores. Figure 5.10 shows that the models with the canonical module have higher rotation accuracy for smaller thresholds on the sprayer category and a higher accuracy overall on the cameras.

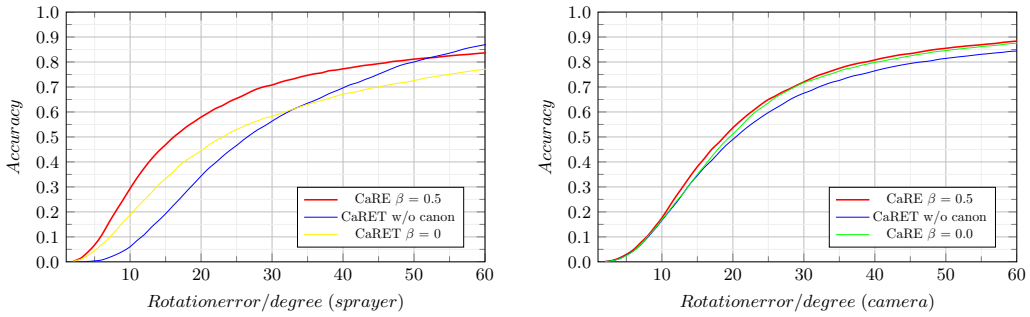


Figure 5.10: Comparison of CaRET to the baseline. CaRET achieves better rotation estimation on the sprayers and cameras.

The default canonical coordinate system of the baseline meshes is fixed for the whole training process, whereas our canonical module allows the network to learn a better canonical coordinate system.

Overall, our pipeline can achieve good results with an AUC over 64.51 for all non-symmetric categories and over 86.59 for our symmetric category. We still need to test our pipeline on real world data, but our experiments show promising results as our predictions are very accurate for the known instances, as we can see in Figure 5.11. Our evaluation shows similar results for both canonical models. We can deduce that the CanonicalLoss is not needed if we have a good initial canonical pose, as is the case for the DeepCPD data set.

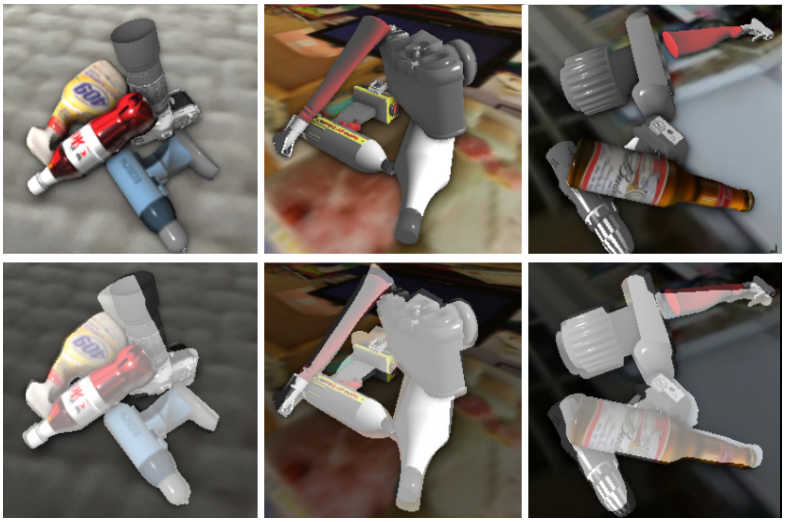


Figure 5.11: Results on the test instances. The top row shows the ground truth, and the bottom row the prediction rendered above the ground truth.

ADD AUC Comparison			
	β=0.5, noise	β=0, no noise	baseline
Drills	70.57	70.85	70.00
Bottles	56.68	58.88	57.68
Cameras	67.45	66.47	64.51
Sprayers	70.07	67.59	66.55
ADD-S AUC Comparison			
	β=0.5, noise	β=0,no noise	baseline
Drills	84.18	84.39	84.27
Bottles	86.71	86.62	86.59
Cameras	84.50	83.64	83.98
Sprayers	84.59	83.76	83.25

Table 5.1: ADD and ADD-s metric for the known instances.

ADD AUC Comparison			
	β=0.5, noise	β=0, no noise	no canon
Drills	51.39	40.21	49.52
Bottles	28.61	22.3	29.26
Cameras	53.89	48.41	50.92
Sprayers	15.14	14.1	15.66
ADD-S AUC Comparison			
	β=0.5, noise	β=0,no noise	no canon
Drills	78.32	72.71	75.13
Bottles	59.98	55.74	60.8
Cameras	80.00	78.15	78.96
Sprayers	50.20	48.27	50.80

Table 5.2: ADD and ADD-s metric for the unknown instances.

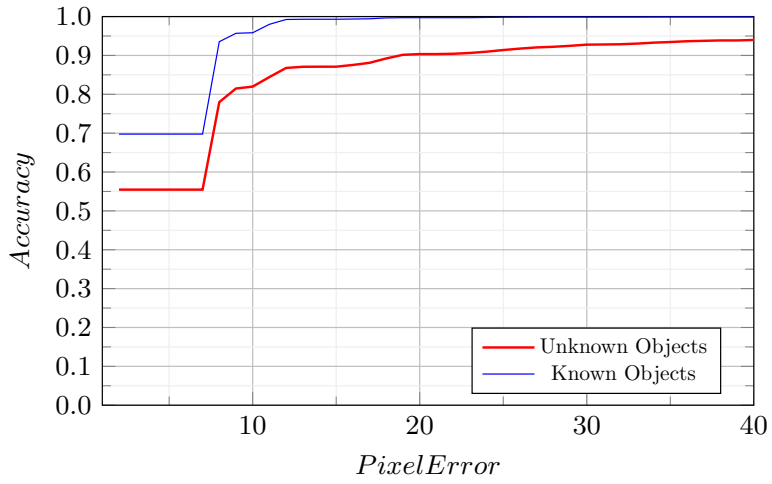


Figure 5.12: Comparison of heatmap accuracy. Accuracy on unknown objects drop especially for smaller tresholds.

Evaluation of Unknown Instances To evaluate how well the CaRET and the baseline models generalize to unknown instances, we create a test data set with unknown instances using the same rendering pipeline used to create the training data set. We limit the number of objects per scene to two. We observe that the models struggle with the new objects and the metrics get a lot worse. While investigating further, we notice that the new meshes combined with the occlusions result in poor heatmaps, increasing the origin offset. Figure 5.12 shows the accuracy for different distance thresholds measured in pixel. The main observation is that the prediction quality drops by 15% for a threshold of eight pixels for unknown objects. The flat part up to a threshold of 8 pixels is caused by resizing our 32×32 heatmap to 256×256. On some rare occasions, the networks confuse the bottle with a sprayer because of their geometrical similarities. If the network cannot detect the spray nozzle, it recognizes the sprayer as a bottle and predicts its origin in the bottle's heatmap. In a few other cases, the prediction of one test drill instance is flipped vertically, i.e., upside down, as seen in Figure 5.13.

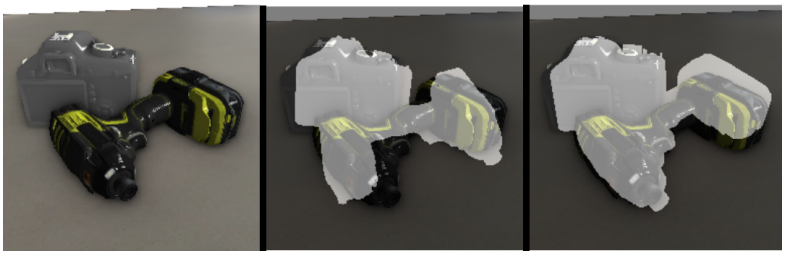


Figure 5.13: Shape differences lead to wrong predictions. The test instance's head is shorter compared to the training examples and the network cannot distinguish between top and bottom.

In Figure 5.14, the models with the canonical module have a higher error rate around 180°, i.e., a flipped rotation, compared to the baseline. Most of the drills have a head that extends over the battery, but this is not the case for one unknown test instance. The network with the canonical module confuses the top with the bottom and predicts a flipped pose. More training samples with varying shapes could help to improve the generalization. Moreover, the problem of predicting the 6D pose for an unknown object only from RGB images is inherently ill-posed. I.e., without depth, estimating the precise scale of an unknown object is not possible. When we assume a correct translation, our predictions get a lot better for the unknown instances. Figure 5.16 shows the estimated rotation rendered with ground truth translation. Here we can observe that our rotation estimation is accurate. Overall our model with the canonical loss performs slightly better for the unknown objects than the model without the canonical loss. Both models align the

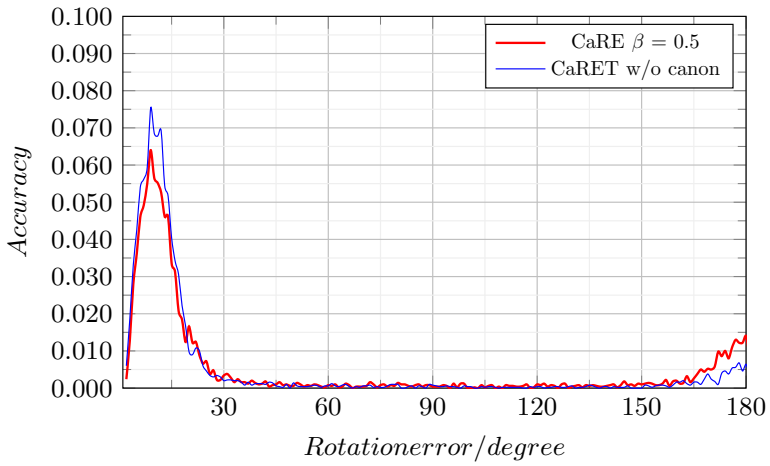


Figure 5.14: Comparison of the rotation error for drills. The model with the canonical module is more likely to flip the prediction.

instances to their learned coordinate system. However, we notice that the network with the canonical loss is less likely to flip the prediction on the drills, as we can see in Figure 5.15. We suspect that the CanonicalLoss helps the canonical module

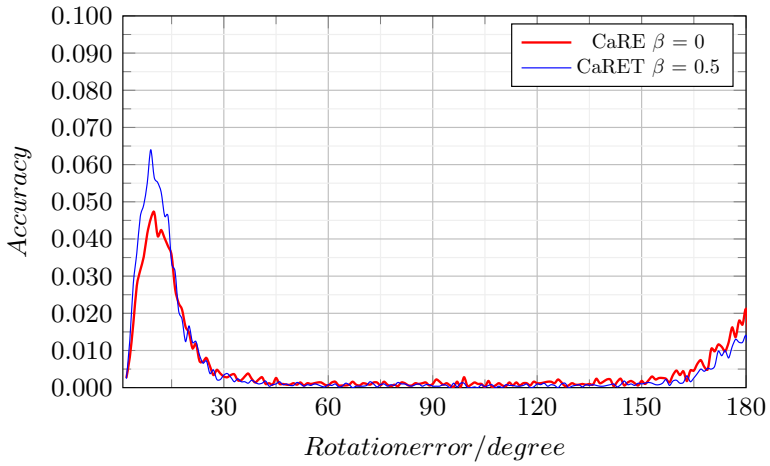


Figure 5.15: Comparison of the rotation error for drills. The model without the canonical loss is more likely to flip the prediction.

learn better discriminative features, which allows for better predictions. A higher β might even help to reduce the flipped predictions even more. To our surprise, the baseline model achieves similar results compared to $\text{model}_{0.5}$ and outperforms the model without the CanonicaLoss. All models perform relatively poorly on the sprayers, which is also the category with the fewest training instances. A larger set of sprayer samples for training could improve the metrics.

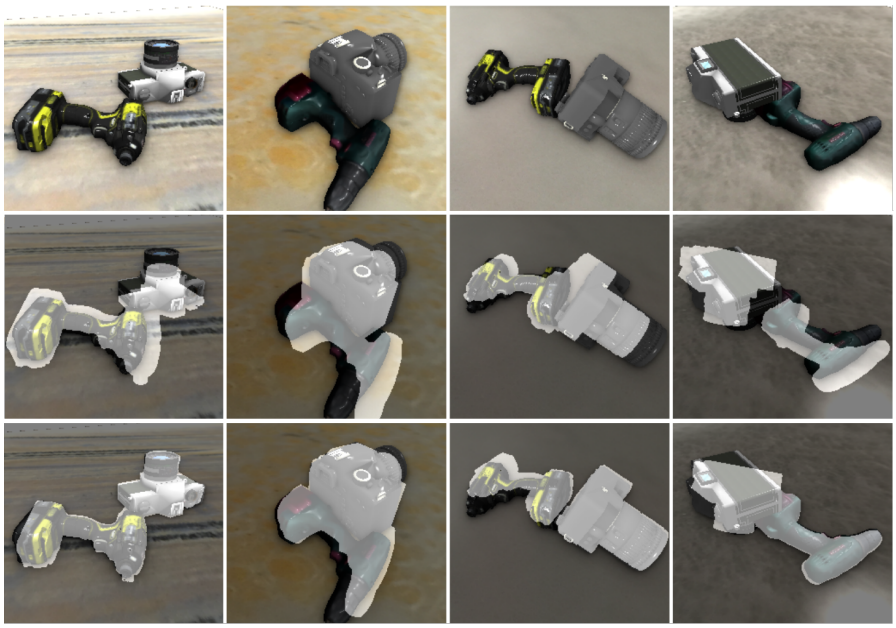


Figure 5.16: The top row is the ground truth, the predictions, rendered on top of the ground truth, are in the middle, and the bottom row assumes a correct translation.

Figure 5.16 shows some acceptable and bad predictions of our network. In the second row, we can see that the network struggles to estimate the translation correctly. The objects are either too big, pointing towards a false depth, or shifted to the left or right, which is the product of a poor heatmap. The third row shows the predictions but assumes a correct translation. The rotation is acceptable when taking into account that the network has not seen this specific instance before. To further improve the predictions, we need to add additional depth information.

6 Conclusion

In this thesis, we developed a category-level pose estimator that learns a category-specific canonical representation. We evaluated it on our own data set, which we created using the Stillleben[3] rendering pipeline. Our CaRET model predicts the translation with using the same approach as PoseCNN[1]. The rotation is split up into the transformation from object to canonical and from canonical to camera coordinates. We optimize it using our CanonicalLoss, which applies the ShapeMatch-Loss to a batch of object instances of the same category, exploiting geometrical similarities. The generated data set is rich in object occlusions and object interactions.

We achieve good results on our cluttered test set with known instances and acceptable results for the unknown objects. Since our network can already estimate the orientation of unknown instances, we can save training time for competitions, where the exact CAD models are known only a short period of time before the competition starts.

Overall, the canonical module shows promising results with improved metrics for most known categories, but only achieves similiar results on the unknown instances compared to our baseline. The module needs further experimentation since it is unclear when to freeze the module. We also need to investigate if more freedom in choosing the origin allows for better predictions. There has also been more research on processing 3D data. Further improvements could be made by replacing the PointNet with a graph neural network that takes the connectivity of verticies into account. Right now, we cannot evaluate on the CAMERA data set because our pipeline cannot handle multiple instances of the same category in the same image. Replacing the RefineNet backbone with an object detector like CornerNet[10], would allow us to solve this problem. For each RoI we can load the corresponding rotational branch and get a good rotation estimation. Since the NOCS approach by [2] belongs to the template matching methods, we can expect better predictions for partly occluded objects. The additional use of depth information could also lead to more accurate results on the unknown instances.

List of Figures

2.1 2×2 Max pooling operation with stride=2. For each 2×2 patch, the max-pooling layer takes the maximum, downsampling the image. . 5

2.2 Comparison of learning rates. A small learning rate converges to a local minimum and does not explore the remaining space. The optimal learning rate explores more space and can find a better solution. A learning rate too big causes the network to oscillate and does not converge. 6

2.3 Heatmaps for origin detection. Each dot in the image represents an object center. The corresponding heatmap encodes the keypoint with an activity blob (Gaussian bell). 8

2.4 Overview of 3D data representations. Existing representations can be broadly categorized in euclidean and non-euclidean data. Image taken from [20]. 9

2.5 Architecture of PoseCNN. Image taken from [1] 11

2.6 Illustration of the camera and object coordinate system. Knowing the center in the image and the distance in the z-direction, we are able to recover the translation T. Image taken from [1] 12

2.7 Motivation and application of continuous representations. A discontinuous representation is harder to learn because both bounds correspond to the same rotation. To improve this Yi Zhou et al. propose a bijection from the representation to the original space. Image taken from [6] 14

2.8 Comaprison of different rotation representations for the task of 3D pose estimation from 2D images. Image was taken from[26] 16

3.1 Image samples from the YCB-Dataset[1]. The data set consists of 92 unique scenes. 23

3.2 Examples from CAMERA[2]. He Wang et al. render the objects in blender and place them on surfaces in real images. 23

4.1 Comparsion of our images to CAMERA. The top row shows three samples taken from CAMERA[2]. Objects are realistically placed but lack shadows, correct lightning and occlusions. The bottom row shows three examples of our data. Our cluttered scenes introduce occlusions and have correct lightning conditions. 26

4.2 Motivation of the canonical coordinate system. Instead of predicting the transformation from object to scene directly, we predict an intermediate coordinate system where all category instances are aligned. During training we have to take the noise transformation into account. 27

4.3 Meshes from the DeepCPD data set. Objects of the same category are similiar in their geometry which allows us to introduce a loss that measures the geometric differences. Image was taken from [4] . 28

4.4 Adjusting the ground truth. During the training process, the network actively learns the canonical coordinate system. Thus we create the ground truth after the networks prediction of the canonical frame. 29

4.5 An origin that lies beyond the image boundaries cannot be detect with our heatmaps. We therefore need to pull it closer to the object. 30

4.6 Pipeline overview. Our pipeline uses a RefineNet architecture to predict a heatmap, z-map, and segmentation. Heatmap and z-map estimate the translation from the canonical frame to the scene. A slightly altered VGG extracts process the features from the backbone. One MLP predicts the transformation to the canonical frame while the other estimates the final rotation. Both use the PointNet features, but only the second MLP uses image features. 32

4.7 Computing the RoI from a segmentation. We find the top-left and bottom-right corner of our segmentation to extract the bounding box. 32

4.8 Category specific rotation branch. The modified VGG learns category specific features which are flattened and passed to an MLP. The MLP processes the image feature vector first and then adds the PointNet features. The output is a rotation in our choosen rotation representation. 34

5.1 Comparison of rotation representations. Higher dimensional rotation representations converge faster. 37

5.2 Comparison of 6D and 9D rotation representations. The 9D representation becomes unstable leading to fluctuation in the canonical loss which in return influences the final scene prediction. 6D converges smoothly and has an overall better performance 38

5.3 Alignment without loss. The model without canonical loss minimizes the CanonicaLoss without us enforcing it. 39

5.4 Generalization to unknown Instances. The graph plots the number of correct predicition using the ADD metric. The model trained with the CanonicalLoss needs noise to achieve similar performance compared to the model trained without. 40

5.5 Comparison of poses. The left side is the original coordinate system and the right side our learned canonical coordinate system. 42

5.6 Comparison of canonical poses. Different betas result in different orientations for the objects. 43

5.7 Evolution of the canonical pose during the trainings process. Starting from the inital pose on the left, the SceneLoss pulls the drills apart, but the CanonicalLoss pulls them back together. 43

5.8 Comparision of the trainings loss between the three networks. Adding the canonical module improves the SceneLoss. The canonical loss further improves the model if object instances are not prealigned. . 44

5.9 Comparison of models with and without noise. Rotational noise in the input makes every model slightly worse except the model that was trained with the canonical loss. 45

5.10 Comparison of CaRET to the baseline. CaRET achieves better rotation estimation on the sprayers and cameras. 46

5.11 Results on the test instances. The top row shows the ground truth, and the bottom row the prediction rendered above the ground truth. 47

5.12 Comparison of heatmap accuracy. Accuracy on unknown objects drop especially for smaller tresholds. 48

5.13 Shape differences lead to wrong predictions. The test instance's head is shorter compared to the training examples and the network cannot distinguish between top and bottom. 49

5.14 Comparison of the rotation error for drills. The model with the canonical module is more likely to flip the prediction. 50

5.15 Comparison of the rotation error for drills. The model without the canonical loss is more likely to flip the prediction. 50

5.16 The top row is the ground truth, the predictions, rendered on top of the ground truth, are in the middle, and the bottom row assumes a correct translation. 51

List of Tables

5.1 ADD and ADD-s metric for the known instances. 47
5.2 ADD and ADD-s metric for the unknown instances. 48

Bibliography

[1] Yu Xiang et al. "Posecnn: a convolutional neural network for 6d object pose estimation in cluttered scenes." In: *Arxiv preprint arxiv:1711.00199* (2017).

[2] He Wang et al. "Normalized object coordinate space for category-level 6d object pose and size estimation." In: *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. 2019, pp. 2642–2651.

[3] Max Schwarz and Sven Behnke. "Stillleben: realistic scene synthesis for deep learning in robotics." In: *2020 ieee international conference on robotics and automation (icra)*. IEEE. 2020, pp. 10502–10508.

[4] Diego Rodriguez, Florian Huber, and Sven Behnke. "Category-level 3d non-rigid registration from single-view rgb images." In: Oct. 2020. DOI: 10.1109/IROS45743.2020.9340878.

[5] Anastasis Kratsios. "The universal approximation property." In: *Annals of mathematics and artificial intelligence* (2021), pp. 1–35.

[6] Yi Zhou et al. "On the continuity of rotation representations in neural networks." In: *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*. 2019, pp. 5745–5753.

[7] Yann LeCun, Y. Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[8] Niall O'Mahony et al. "Deep learning vs. traditional computer vision." In: *Science and information conference*. Springer. 2019, pp. 128–144.

[9] Tomasz Szandała. "Review and comparison of commonly used activation functions for deep neural networks." In: *Bio-inspired neurocomputing*. Springer, 2021, pp. 203–224.

[10] Hei Law and Jia Deng. "Cornernet: detecting objects as paired keypoints." In: *Proceedings of the european conference on computer vision (eccv)*. 2018, pp. 734–750.

[11] Adam Paszke et al. "Pytorch: an imperative style, high-performance deep learning library." In: *Arxiv preprint arxiv:1912.01703* (2019).

[12] Ilya Loshchilov and Frank Hutter. "Sgdr: stochastic gradient descent with warm restarts." In: *Arxiv preprint arxiv:1608.03983* (2016).

[13] *Pytorch autograd*. https://pytorch.org/tutorials/beginner/basics/autograd_tutorial.html. Accessed: 2021-05-11.

[14] Diederik P Kingma and Jimmy Ba. "Adam: a method for stochastic optimization." In: *Arxiv preprint arxiv:1412.6980* (2014).

[15] *Pytorch optimizers and schedulers*. `https://pytorch.org/docs/stable/optim.html`. Accessed: 2021-05-11.

[16] Leslie N Smith. "Cyclical learning rates for training neural networks." In: *2017 ieee winter conference on applications of computer vision (wacv)*. IEEE. 2017, pp. 464–472.

[17] Kaiming He et al. "Mask r-cnn." In: *Proceedings of the ieee international conference on computer vision*. 2017, pp. 2961–2969.

[18] Alejandro Newell, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation." In: vol. 9912. Oct. 2016, pp. 483–499. ISBN: 978-3-319-46483-1. DOI: 10.1007/978-3-319-46484-8_29.

[19] A Specification. "The opengl." In: (1992).

[20] Eman Ahmed et al. "A survey on deep learning advances on different 3d data representations." In: *Arxiv preprint arxiv:1808.01462* (2018).

[21] Yin Zhou and Oncel Tuzel. "Voxelnet: end-to-end learning for point cloud based 3d object detection." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2018, pp. 4490–4499.

[22] Charles R Qi et al. "Pointnet: deep learning on point sets for 3d classification and segmentation." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2017, pp. 652–660.

[23] Guosheng Lin et al. "Refinenet: multi-path refinement networks for high-resolution semantic segmentation." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2017, pp. 1925–1934.

[24] Andy Zeng et al. "Multi-view self-supervised deep learning for 6d pose estimation in the amazon picking challenge." In: *2017 ieee international conference on robotics and automation (icra)*. IEEE. 2017, pp. 1386–1383.

[25] Stefan Hinterstoisser et al. "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes." In: *2011 international conference on computer vision*. IEEE. 2011, pp. 858–865.

[26] Jake Levinson et al. "An analysis of svd for deep rotation estimation." In: *Arxiv preprint arxiv:2006.14616* (2020).

[27] Charles R Qi et al. "Pointnet++ deep hierarchical feature learning on point sets in a metric space." In: *Proceedings of the 31st international conference on neural information processing systems*. 2017, pp. 5105–5114.

[28] Yi Li et al. "Deepim: deep iterative matching for 6d pose estimation." In: *Proceedings of the european conference on computer vision (eccv)*. 2018, pp. 683–698.

[29] Yannick Bukschat and Marcus Vetter. "Efficientpose–an efficient, accurate and scalable end-to-end 6d multi object pose estimation approach." In: *Arxiv preprint arxiv:2011.04307* (2020).

[30] Alvaro Collet, Manuel Martinez, and Siddhartha S Srinivasa. "The moped framework: object recognition and pose estimation for manipulation." In: *The international journal of robotics research* 30.10 (2011), pp. 1284–1306.

[31] Saurabh Gupta et al. "Aligning 3d models to rgb-d images of cluttered scenes." In: June 2015, pp. 4731–4740. DOI: 10.1109/CVPR.2015.7299105.

[32] Michael Hödlmoser et al. "Classification and pose estimation of vehicles in videos by 3d modeling within discrete-continuous optimization." In: *2012 second international conference on 3d imaging, modeling, processing, visualization & transmission*. IEEE. 2012, pp. 198–205.

[33] Daniel P Huttenlocher, Gregory A. Klanderman, and William J Rucklidge. "Comparing images using the hausdorff distance." In: *Ieee transactions on pattern analysis and machine intelligence* 15.9 (1993), pp. 850–863.

[34] Paul J Besl and Neil D McKay. "Method for registration of 3-d shapes." In: *Sensor fusion iv: control paradigms and data structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–606.

[35] Catherine Capellen, Max Schwarz, and Sven Behnke. "Convposecnn: dense convolutional 6d object pose estimation." In: *Arxiv preprint arxiv:1912.07333* (2019).

[36] Eric Brachmann et al. "Learning 6d object pose estimation using 3d object coordinates." In: *European conference on computer vision*. Springer. 2014, pp. 536–551.

[37] Fabian Manhardt et al. "Explaining the ambiguity of object detection and 6d pose from visual data." In: *Proceedings of the ieee/cvf international conference on computer vision*. 2019, pp. 6841–6850.

[38] Shuran Song and Jianxiong Xiao. "Sliding shapes for 3d object detection in depth images." In: *European conference on computer vision*. Springer. 2014, pp. 634–651.

[39] Shuran Song and Jianxiong Xiao. "Deep sliding shapes for amodal 3d object detection in rgb-d images." In: *Proceedings of the ieee conference on computer vision and pattern recognition*. 2016, pp. 808–816.

[40] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: convolutional networks for biomedical image segmentation." In: *International conference on medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[41] H. Noh, S. Hong, and B. Han. "Learning deconvolution network for semantic segmentation." In: *2015 ieee international conference on computer vision (iccv)*. 2015, pp. 1520–1528. DOI: 10.1109/ICCV.2015.178.

[42] Evan Shelhamer, Jonathon Long, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In: *Ieee transactions on pattern analysis and machine intelligence* 39 (May 2016), pp. 1–1. DOI: 10.1109/TPAMI.2016.2572683.

[43] Weiwei Sun et al. "Canonical capsules: unsupervised capsules in canonical pose." In: *Arxiv preprint arxiv:2012.04718* (2020).

[44] *Coco dataset*. https://cocodataset.org/#home. Accessed: 2021-05-11.

[45] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *Arxiv preprint arxiv:1409.1556* (2014).

[46] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world." In: *2017 ieee/rsj international conference on intelligent robots and systems (iros)*. IEEE. 2017, pp. 23–30.

[47] Angel Chang et al. "Shapenet: an information-rich 3d model repository." In: (Dec. 2015).

[48] Chen Wang et al. "6-pack: category-level 6d pose tracker with anchor-based keypoints." In: May 2020, pp. 10059–10066. DOI: 10.1109/ICRA40945.2020.9196679.

[49] Alexander D'Amour et al. "Underspecification presents challenges for credibility in modern machine learning." In: *Arxiv preprint arxiv:2011.03395* (2020).

[50] M. Strese et al. "A haptic texture database for tool-mediated texture recognition and classification." In: *2014 ieee international symposium on haptic, audio and visual environments and games (have) proceedings*. 2014, pp. 118–123. DOI: 10.1109/HAVE.2014.6954342.

[51] *Messy and clean room dataset*. https://www.kaggle.com/cdawn1/messy-vs-clean-room. Accessed: 2021-05-11.

[52] Sergey I Nikolenko. "Synthetic data for deep learning." In: *Arxiv preprint arxiv:1909.11512* (2019).