CloudAAE: Learning 6D Object Pose Regression with On-line Data Synthesis on Point Clouds

Ge Gao[1], Mikko Lauri[1], Xiaolin Hu[2], Jianwei Zhang[1] and Simone Frintrop[1]

Abstract— **It is often desired to train 6D pose estimation systems on synthetic data because manual annotation is expensive. However, due to the large domain gap between the synthetic and real images, synthesizing color images is expensive. In contrast, this domain gap is considerably smaller and easier to fill for depth information. In this work, we present a system that regresses 6D object pose from depth information represented by point clouds, and a lightweight data synthesis pipeline that creates synthetic point cloud segments for training. We use an augmented autoencoder (AAE) for learning a latent code that encodes 6D object pose information for pose regression. The data synthesis pipeline only requires texture-less 3D object models and desired viewpoints, and it is cheap in terms of both time and hardware storage. Our data synthesis process is up to three orders of magnitude faster than commonly applied approaches that render RGB image data. We show the effectiveness of our system on the LineMOD, LineMOD Occlusion, and YCB Video datasets. The implementation of our system is available at: `https://github.com/GeeeG/CloudAAE`.**

I. Introduction

A 6 degrees of freedom (6D) pose is a 3D rotation and 3D transformation between a local object coordinate and a camera or robot coordinate. Knowing the 6D poses of objects is important for applications such as robotic grasping and manipulation [1]. Estimating 6D object pose from images is challenging due to occlusion, background clutter, and different illumination conditions. According to the recent benchmark challenge for 6D object pose estimation (BOP) [2], the performance of deep learning methods has improved compared to the previous year. Deep learning-based methods benefit from large amounts of high quality training data. Although there are tools and approaches for annotating the 6D object pose on real images [3], [4], the annotation for the 6D object pose is expensive. Moreover, the accuracy of manual labels can not always be guaranteed [4].

Creating synthetic data eliminates the need for manual labeling and can guarantee the accuracy of labels. The difference in appearance between real and synthetic data is called the domain gap. The domain gap between real and synthetic RGB images is often large. In the most recent BOP challenge, the main reason for the performance boost of deep learning-based methods is having additional photo-realistic synthetic images for training [2]. These images are created using a physically-based renderer (PBR) and present a much smaller domain gap, compared to naive approaches such as "render & paste" [2]. Despite being effective, PBR is expensive in the terms of time and hardware storage. This high cost makes it challenging to scale PBR up to a large number of objects, which is often desired in robotic applications. Methods using only depth information are more robust in the presence of the domain gap [5]. This indicates that if synthesizing data with only depth information, the domain gap is potentially much smaller and easier to fill. This opens up the possibility to create a lightweight data synthesis pipeline using only depth information.

In this work, we propose a point cloud based 6D pose estimator and a lightweight data synthesis pipeline. We propose to use an implicit approach during the pose inference stage. We achieve this by adapting the augmented autoencoder (AAE) proposed by Sundermeyer et al. [6] for a point cloud based system. By using an AAE, we are able to control which properties the latent code encodes and which properties are ignored [6]. This is achieved by applying augmentations to the input, and the encoding becomes invariant to those augmentations. We focus on learning a latent code that encodes the 6D pose information. The task of the AAE is to reconstruct a point cloud segment in the desired 6D pose. Moreover, this segment is noise and occlusion free. In this way, the latent code contains the necessary information for regressing the object pose. Using the latent code as the input, we use two separate networks for regressing 3D rotation and 3D translation, as suggested in [7], [8]. The on-line data synthesis pipeline requires a texture-less 3D object model and the desired viewpoint as the input. The computational cost is low.

Our contributions are:

- We present a new framework for regressing the 6D object pose from point cloud segments. An point cloud based augmented autoencoder is used to learn a latent code that encodes object pose information. This code is used for regressing the 6D object pose.
- We present a point cloud based lightweight data synthesis pipeline for generating training data. Compared to existing RGB based data synthesis systems, the cost of ours is lower in the sense of time and hardware storage.

We show the effectiveness of the combination of our data synthesis pipeline and pose estimation system on three datasets. When using only synthetic training data, our model achieves state-of-the-art performance among other synthetic trained methods on the LineMOD dataset [9]. Our cheap

*This work is funded by the German Science Foundation (DFG) in project Crossmodal Learning, TRR 169

[1]Department of Informatics, University of Hamburg, Germany {gao,lauri,zhang}@informatik.uni-hamburg.de frintrop@informatik.uni-hamburg.de

[2]State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, China {xlhu}@mails.tsinghua.edu.cn

synthetic point cloud data can replace costly render based synthetic data for training systems using depth for pose inference.

II. Related Work

We review deep learning 6D pose estimators on point clouds, unsupervised learning of 6D pose, as well as how training data is synthesized.

Deep learning 6D pose estimation on point clouds. Frustum PointNets [10] uses point cloud segments for pose estimation. They use PointNet [11] for processing point clouds. The rotation estimation is formulated as a classification problem, and the result for one angle is reported. Densefusion [12] uses PointNet to extract feature vectors from point cloud segments, and a convolutional neural network (CNN) to extract features from the corresponding color information. Those features are combined for regressing the 6D object pose. PVN3D [13] adapts the feature extraction pipeline from DenseFusion, and uses a 3D Keypoint detection and Hough voting scheme for 6D pose estimation. CloudPose [8] uses a PointNet like structure to extract features from point clouds, and directly regress to 3D rotation and 3D translation with two separate networks. Our method is the most similar to CloudPose, as we regress to 6D object poses from point clouds. However, rather than directly regressing to 6D pose from point clouds, we use the latent code from an AAE as the input to pose regressors.

Unsupervised learning of 6D pose. The augmented autoencoder is proposed in [6]. It is a variant of the Denoising Autoencoder [14]. The authors use 2D bounding boxes for translation estimation, and use the AAE for 3D rotation estimation. For 3D rotation, rather than explicitly mapping from an input image to a pose label, the AAE learns implicit codes of object orientations in a latent space. A codebook of latent codes is created off-line, and they use a nearest neighbor search to compare a test code within the codebook. This approach is improved by [15] by adding edge priors. Our 6D pose estimation pipeline adapts the AAE concept [6] for point clouds. There are three main differences between our approach and [6]. First, our approach does not require creating off-line codebooks. Second, we use the latent code from the AAE for both 3D translation and 3D rotation estimation. Third, they use 2D color image as input while ours uses point clouds.

Data synthesis methods for 6D pose estimation. Existing data synthesis methods outputs synthetic color images. To create synthetic data, the viewpoint and the appearance from that viewpoint must be determined. Object appearance is obtained from either textured 3D object models or real world training datasets. Viewpoints are selected using some simple or dataset-based heuristics. One common approach is to define spheres around the textured 3D object model with fixed radii, and sample viewpoints from a hemisphere that covers the upper part of the object model [9], [16], or from the full sphere [6], [15]. The object model is rendered in the target viewpoint onto a plain or random background. The advantage of this method is that it provides a good coverage for 3D rotation. The limitation is that using fixed radii puts constraints on the coverage of 3D translation.

The object appearance and viewpoint can also be directly obtained by segmenting foreground objects from an existing training set [17], [18], [19]. More data augmentation, such as background noise, can be applied to the rendered image for reducing the domain gap [20]. However, due to the domain gap, this strategy tends to be insufficient by itself for the 6D object pose estimation task and causes performance drop [5]. Moreover, this approach also relies on having the annotated real training data.

Some approaches synthesize data with photo-realistic appearance and physically plausible object poses [20], [21], [22]. This approach requires a render to create object appearances with realistic lighting and reflection, and physics simulators to provide physically plausible poses. Rendering is computationally expensive, and the data created off-line requires large amounts of hardware storage. Compared to existing approaches, ours is computationally lightweight, and does not need textured 3D object models or segmented foreground objects from real-world training data. Another difference is that the existing approaches mostly conduct off-line data generation while ours can be used on-line.

III. System Overview

We propose a data synthesis pipeline and an augmented autoencoder (AAE) based 6D pose estimation system, referred to as CloudAAE. CloudAAE is a multi-class system and we use the same system to predict poses for objects from different classes. Figure 1 shows an overview of the proposed system. Given a known object represented by a set of points in the camera coordinate C, the aim of a 6D pose estimation system is to find the translation $\mathbf{t}$ and rotation R that describes the transformation from the object coordinate system O to C. The data synthesis pipeline creates a point cloud segment P^C_{occ}, and P^C_{occ} is normalized by removing its coordinate mean $\mu_{\mathbf{P}}$ before further steps. We denote the normalized P^C_{occ} as P^N_{occ}.

By using the data synthesis pipeline, we apply random noise and occlusion to the input point cloud segment to add variance in the training samples. The AAE reconstructs a noise and occlusion free object segment at the desired 6D pose. Hence, the latent vector encodes the 6D object pose information, and it is used for 6D pose regression.

IV. Data synthesis pipeline

The inputs for the data synthesis pipeline are a 3D object model and a 6D pose. We use 3D object models represented by point clouds. A 3D rotation R and a 3D translation $\mathbf{t}$ are drawn from a desired pose distribution. The desired pose distribution can be the poses in a training set [17], [19]. It can also be a pool of poses covering the same distribution as the poses in a training set [23], [7], [24]. In this work, if the train dataset contains a large amount of poses, we use them as the 6D pose for data synthesis. Otherwise, we draw poses from the distribution of train poses.

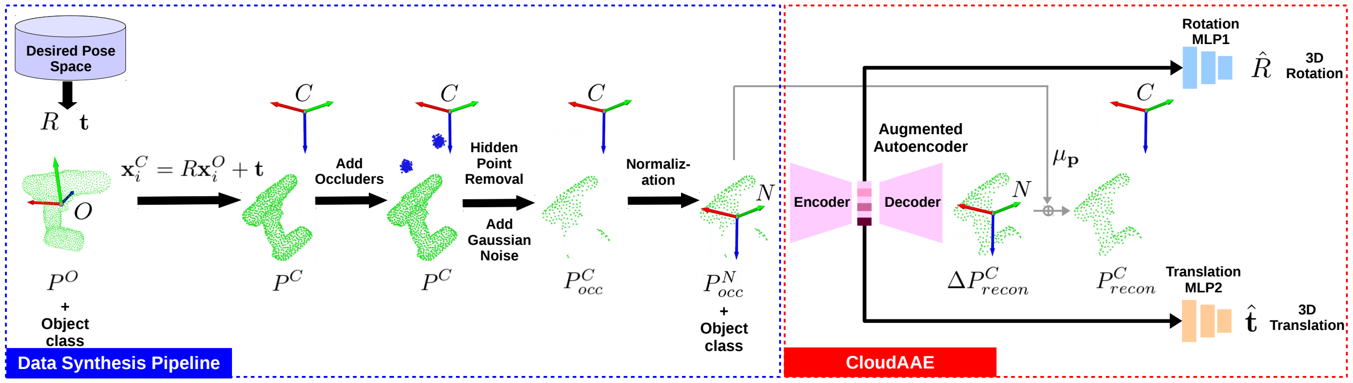


Fig. 1. Left, Data synthesis pipeline. The input for our on-line training pipeline is a 3D object model P^O with class information, and the desired 3D rotation R and translation $\mathbf{t}$. P^O is at object coordinate O. With R and $\mathbf{t}$, the object model is first transformed to the camera coordinate C. The transformed model is denoted by P^C. Spherical occluders S (denoted in blue) are added between C and P^C. Hidden point removal is applied to $P^C \cup S$ to remove points in P^C and are not visible from C. Zero-mean Gaussian noise is added to each remaining point to introduce variance. The final point set is P^C_{occ}. P^C_{occ} is normalized into P^N_{occ}, by subtracting the mean $\mu_\mathbf{p}$ of P^C_{occ}. Right, CloudAAE. The input is P^N_{occ} and the corresponding class information. The expected ΔP^C_{recon} is a noise and occlusion free segment. By adding $\mu_\mathbf{p}$ to ΔP^C_{recon}, we obtain P^C_{recon} at the desired 6D pose. Meanwhile, the latent code is used with two separate 3-layer multi-layer perceptrons (MLP1 and MLP2) for regressing 3D rotation $\hat{R}$ and 3D translation $\hat{\mathbf{t}}$.

Given an object model represented by the set of points $P^O = \{\mathbf{x}_i^O \in \mathbb{R}^3 \mid i = 1, 2, \ldots, n\}$, the model is transformed from its object coordinate system O to the camera coordinate system C by

$$\mathbf{x}_i^C = R\mathbf{x}_i^O + \mathbf{t}. \tag{1}$$

$P^C = \left\{\mathbf{x}_i^C \in \mathbb{R}^3 \middle| i = 1, 2, \ldots, n\right\}$ is the transformed model, where $\mathbf{x}_i^C$ is the ith point. To simulate occlusions, we randomly generate a set of points S as the spherical occluders (denoted in blue in Figure 1) between the camera origin C and P. We remove points in $P^C \cup S$ that are not visible from C by applying hidden point removal (HPR) [25]. From the set of visible points, we remove the points belonging to S. To add variance to the training sample, we add zero-mean Gaussian noise with standard deviation σ to each remaining point. In all of our experiments, we use $\sigma = 1.3mm$. The final resulting object segment is $P^C_{occ} = \left\{\mathbf{x}_i \in \mathbb{R}^3 \mid i = 1, \ldots, m\right\}$.

V. CloudAAE: Augmented Autoencoder based 6D Pose Estimation

In this section, we introduce our augmented autoencoder-based system CloudAAE, which computes the 6D pose of objects from point cloud data.

A. *Network structure*

CloudAAE contains an augmented autoencoder, and the 6D pose regressors. We adapted the idea of AAE proposed in [6] for point clouds, and use an adapted version of the dynamic graph CNN (DGCNN) [26] as the encoder. We additionally add two networks for regressing the 6D poses.

Augmented autoencoder (AAE): Figure 2 illustrates the architecture of our point cloud based AAE. The AAE consists of an encoder and a decoder. The 3D coordinate of each point in P^N_{occ} is concatenated with one-hot class information. This is used as the input to the encoder. The encoder computes a latent vector, and the desired output from the decoder is a noise and occlusion free segment in the 6D pose defined by R and $\mathbf{t}$. The input is of dimension $n \times (3 + c)$, in which 3 represents 3D coordinates, and c is the total number of classes.

Our encoder is an adapted version of DGCNN, which is a deep network processing unordered point sets. Assume $Q = \{\mathbf{q}_i \in \mathbb{R}^p \mid i = 1, \ldots, m\}$ is a point set. $\mathbb{R}^p$ can be a 3D space or an arbitrary feature space. For each point $\mathbf{q}_i$, a k-nearest neighbor graph is calculated. In all our experiments, we use $k = 10$. The graph contains directed edges $(i, j_{i1}), \ldots, (i, j_{ik})$, in which $\mathbf{q}_{j_{i1}}, \ldots, \mathbf{q}_{j_{ik}}$ are the k closest points to $\mathbf{q}_i$. For the edge $\mathbf{e}_{ij}$, an edge feature $\left[\mathbf{q}_i, \quad (\mathbf{q}_j - \mathbf{q}_i)\right]^T$ is calculated.

The edge features are processed by an edge convolution operation (EdgeConv), which contains an edge function and an aggregation operation [26]. For the edge function, we use an MLP layer with the shared weights for each edge feature. We use average pooling as the aggregation operation. This EdgeConv is repeated, calculating the nearest neighbor graph for the feature vectors of the first shared MLP layer, as well as for the subsequent layers. Finally, the edge features from each EdgeConv are concatenated and processed with an MLP layer. This concatenation is illustrated with skip connections in Figure 2. A 1024-dimensional feature vector is learned for each point. These features are average-pooled to obtain a global representation of the input point cloud segment. We use four EdgeConv layers on edge features and one EdgeConv layer on the concatenated edge feature. The encoder outputs a latent code with the dimension of 1024. The decoder contains 3 fully connected layers with dimensions of 1024, 1024 and $n \times 3$.

6D pose regressors. Since the decoder is able to reconstruct the segment in the same 6D pose from the latent code, the latent code contains the object pose information. Hence, the latent code is used as the input to two networks for regressing 3D rotation and 3D translation, respectively. Each network contains three MLP layers with dimensions 512, 256, and 3, respectively.

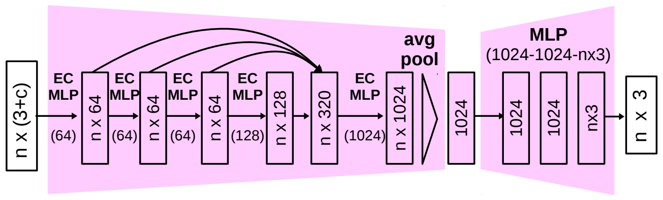


Fig. 2. Point cloud based Augmented Autoencoder. The numbers of neurons in EC-MLP and MLP layers are indicated in parentheses. EC is short for EdgeConv. Dimensions of intermediate features are indicated without parentheses. Skip connections denote the concatenation of edge features. For the encoder, a 1024-dimensional feature vector for each point with its local neighbors is learned with shared weights. An average pooling layer aggregates the individual features into a 1024-dimensional latent code. Finally, three fully-connected layers output a noise and occlusion free point cloud segment.

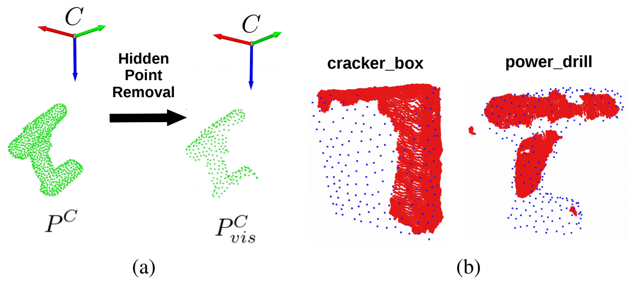


Fig. 3. Illustration of the learning target and the output of AAE. (a) The learning target P^C_{vis}. (b) Examples of AAE output for objects in the YCB video dataset during testing. The input to the AAE is denoted in red, and the reconstructed noise and occlusion free point cloud segment is denoted in blue.

B. Loss function

We normalize P^C_{occ} by removing its mean $\mu_\mathbf{p}$ before inputting to the AAE (Figure 1). The output of AAE is the residual 3D coordinates of the reconstructed point cloud segment. The output of translation prediction is the residual of translation. The full 3D coordinates and 3D translation are obtained by adding back $\mu_\mathbf{p}$.

Augmented autoencoder (AAE). Assuming the AAE outputs residual 3D coordinates ΔP^C_{recon}, the full 3D coordinate is $P^C_{recon} = \Delta P^C_{recon} + \mu_\mathbf{p}$. The learning target P^C_{vis} is obtained by applying HPR to P^C without introducing any occluders, as illustrated in Figure 3(a). Chamfer distance is used to measure the difference between P^C_{recon} and P^C_{vis},

$$l_{CD}(P^C_{recon}, P^C_{vis}) = \frac{1}{m}\sum_{\mathbf{x}\in P^C_{recon}} \min_{\mathbf{y}\in P^C_{vis}} \|\mathbf{x}-\mathbf{y}\|_2 + \frac{1}{n}\sum_{\mathbf{y}\in P^C_{vis}} \min_{\mathbf{x}\in P^C_{recon}} \|\mathbf{y}-\mathbf{x}\|_2. \quad (2)$$

The number of points in P^C_{vis} and P^C_{recon} are denoted by m and n, respectively. The desired P^C_{recon} is a denoised, unoccluded object segment. Figure 3(b) shows examples of P^C_{recon} output by a trained AAE.

6D pose regressors. For rotation regression, the axis-angle representation is used as the regression target. An axis-angle is a vector $\mathbf{r} \in \mathbb{R}^3$ that represents a rotation of $\theta = \|\mathbf{r}\|_2$ radians around the unit vector $\frac{\mathbf{r}}{\|\mathbf{r}\|_2}$. Geodesic distance is used as the loss function for measuring the distance between prediction $\hat{\mathbf{r}}$ and ground truth $\mathbf{r}$ [27]. With $\hat{R}$ and R being the corresponding rotation matrices for $\hat{\mathbf{r}}$ and $\mathbf{r}$ respectively, the *rotation loss function* $l_r(\hat{\mathbf{r}}, \mathbf{r})$ is defined as

$$l_r(\hat{\mathbf{r}}, \mathbf{r}) = \arccos\left(\frac{\text{trace}(\hat{R}R^T) - 1}{2}\right). \quad (3)$$

The regression target for translation is the residual of translation. Given $\widehat{\Delta\mathbf{t}}$ as the translation residual, full translation prediction is $\hat{\mathbf{t}} = \widehat{\Delta\mathbf{t}} + \mu_\mathbf{p}$. With $\mathbf{t}$ being the ground truth translation, the *translation loss function* $l_t(\hat{\mathbf{t}}, \mathbf{t})$ is measured with $L2$-norm $l_t(\hat{\mathbf{t}}, \mathbf{t}) = \left\|\mathbf{t} - \hat{\mathbf{t}}\right\|_2$.

Total loss function. The total loss l_{total} used for training the AAE and pose regressors is the combination of the reconstruction and the pose loss:

$$l_{total} = \alpha l_{CD} + \beta l_t + l_r, \quad (4)$$

where α and β are scaling factors.

C. Testing phase

In the testing phase, the trained CloudAAE is used for estimating 6D object poses. If the input is an RGBD image, semantic segmentation is used to obtain the object segment and the corresponding class information. After segmenting the depth image, the point cloud is obtained using the depth value and the camera intrinsic parameters. The point cloud segment is normalized and concatenated with one-hot class information. This is used as the CloudAAE input. As semantic segmentation is a well-studied topic [28], we assume the object segment and class information is provided by an off-the-shelf method. After obtaining the pose estimates, we further refine the poses with Iterative Closest Point (ICP).

VI. Experiments

We compare our results to the state-of-the-art methods on public datasets. We also investigate the impact of the amount of synthetic data used for training, as well as different sizes for the latent code in AAE.

A. Datasets and experiment setup

We consider three datasets, the LineMOD dataset (LM) [9], LineMOD Occlusion (LMO) [29] and YCB video dataset (YCBV) [4]. Depending on the applicability of the methods, for each dataset, we compare to a subset of state-of-the-art methods SSD-6D [30], EEPG-AAE [15], CloudPose [8], PVNet [31], PoseCNN [4] (PC), DenseFusion [12] (DF), PVN3D [13] and PointVoteNet [32].

LM [9] contains 13 objects from daily scenarios in 13 videos. Although the target objects are not occluded, this dataset is very challenging for depth based methods due to noise in the depth data [15]. LMO [29] contains 8 objects from the LM dataset. The objects belong to one video sequence, and have a high level of occlusion. YCBV [4] contains 21 objects from YCB object set [33] in 92 videos. It contains many objects with different degrees of rotational symmetry, as well as occlusions. It provides both real and synthetic training data. Compared to the other two datasets, the quality of its depth information is good.

We use the 3D models in point clouds provided by the datasets for the on-line data synthesis. Tests are conducted with real test images from the official test split. LM provides approximately 200 training poses for each class, which is a small amount for generating synthetic data. To obtain more 6D pose for training, we first calculate a kernel density estimate on the training set poses. Then we draw $100k$ 6D poses from the distribution for each object class for data synthesis. For YCBV, we use the $80k$ 6D poses from its synthetic training set as the 6D poses for data synthesis.

For training, we use Adam optimizer with learning rate 0.0008. The batch size is 128. The number of points of the input point cloud segments is $n = 256$. Batch normalization is applied to all layers and no dropout is used. For 6D pose regression, we use $\alpha = 1000$ and $\beta = 10$, which is given by the ratio between the expected errors of the reconstruction, translation and rotation at the end of the training [8]. Standard derivation for zero-mean Gaussian noise is $1.3mm$. For ICP refinement, we use the simple Point-to-Point registration provided by Open3D [34] and refine for 10 iterations. The initial search radius is 0.01 meter and it is reduced by 10% after each iteration. When training with synthetic data, both CloudPose [8] and ours use the proposed data synthesis pipeline. For testing, our method, CloudPose and DenseFusion [12] require an off-the-shelf semantic segmentation method. For LM and LMO, both our method and CloudPose use the test object segmentation provided by the corresponding dataset. For YCBV, ours, CloudPose and DenseFusion use the object segmentation provided by PoseCNN [4]. When training with additional real data, we first generate the synthetic segments (P^O_{occ} in Fig. 1) off-line and use them with the real data for training.

B. Evaluation Metrics

The average distance (ADD) of model points and the average distance for a rotationally symmetric object (ADD-S) proposed in [9] are used as evaluation metrics. With a set M with m points representing a 3D model, the ADD is defined as:

$$\text{ADD} = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{M}} \left\| (R\mathbf{x} + \mathbf{t}) - (\hat{R}\mathbf{x} + \hat{\mathbf{t}}) \right\|_2, \tag{5}$$

in which R and $\mathbf{t}$ are the ground truth rotation and translation, and $\hat{R}$ and $\hat{\mathbf{t}}$ are the estimated rotation and translation. ADD-S is computed using closest point distance. It is a distance measure that considers possible pose ambiguities caused by object rotational symmetry:

$$\text{ADD-S} = \frac{1}{m} \sum_{\mathbf{x}_1 \in \mathcal{M}} \min_{\mathbf{x}_2 \in \mathcal{M}} \left\| (R\mathbf{x}_1 + \mathbf{t}) - (\hat{R}\mathbf{x}_2 + \hat{\mathbf{t}}) \right\|_2. \tag{6}$$

For YCBV, a 6D pose estimate is considered to be correct if ADD and ADD-S are smaller than a given threshold 0.1m. The area under error threshold-accuracy curve (AUC) for ADD and ADD-S is calculated with maximum threshold 0.1m. For LM and LMO Occlusion, a pose is considered correct if the error is less than 10% of the maximum diameter of the target object [9].

TABLE I

ESTIMATION ACCURACY OF CLOUDAAE USING DIFFERENT TRAINING DATA.

	Real	Synthetic	Both
LM	86.7	92.5	95.5
LMO	54.9	63.2	66.1
YCBV	-	93.5	93.6

TABLE II

ESTIMATION ACCURACY ON LM, LMO, AND YCBV USING SYNTHETIC TRAINING DATA. OUR METHOD IS CLOUDAAE.

	RGB	D			RGBD
	EEGP-AAE +ICP [15]	CloudPose +ICP [8]	CloudAAE	CloudAAE +ICP	SSD-6D +ICP [30]
LM[*]	89.2	75.2	82.1	**92.5**	90.9
LMO	-	44.2	57.1	**63.2**	-
YCBV[†]	-	93.0	71.0	**93.5**	-

[*] For LM and LMO, ADD-S metric is used for E.box and glue, and others with ADD by convention [30].
[†] For YCBV, ADD-S metric is used for all objects by convention [13].

C. Comparison of prediction accuracy

We first show the effectiveness of our synthetic data by comparing the accuracy of CloudAAE trained with different training data (a). Then, we compare our approach with the state of the art, first on synthetic data and then when using additional real training data (b).

a) Effectiveness of our synthetic data: We train CloudAAE with only real, only synthetic, and both real and synthetic data on all three datasets. The real training data is taken from the corresponding datasets. The synthetic training data is generated with our data generation pipeline. The test performance is shown in Table I. With only synthetic training data, CloudAAE is able to achieve better performance compared to using real training data on LM and LMO. Performance is further improved on all datasets when combining real and synthetic training data. This shows the usefulness of our data synthesis pipeline.

b) Comparison with state-of-the-art methods: In Table II, we compare our CloudAAE method with other state of the art approaches, all trained on synthetic data only. On LM, we outperform the state-of-the-art performance using only depth for pose inference. This shows the effectiveness of the combination of our data synthesis pipeline and CloudAAE. Both CloudAAE and CloudPose use depth for pose inference, but CloudAAE generalizes better to real test data. This shows learning a latent code that encodes pose information can improve system robustness. Using depth for pose inference, our method CloudAAE outperforms the RGBD based method SSD-6D. On LMO, ours outperforms CloudPose by a large margin. On YCBV, our method is slightly better compared to CloudPose. To the best of our knowledge, we are the first to report results using only synthetic training data on this dataset.

TABLE III

ESTIMATION ACCURACY ON LM, LMO, AND YCBV USING REAL AND SYNTHETIC TRAINING DATA. OUR METHOD IS CLOUDAAE.

	RGB		D			RGBD		
	PC [4]	PVNet [31]	PointVotNet +ICP [32]	CloudAAE	CloudAAE +ICP	PC +ICP [4]	DF [12]	PVN3D +ICP [13]
LM	-	-	-	86.8	95.5	-	-	99.4
LMO	24.9	47.3	52.6	58.9	66.1	78.0	-	-
YCBV	-	-	-	-	94.0	-	93.2	96.1

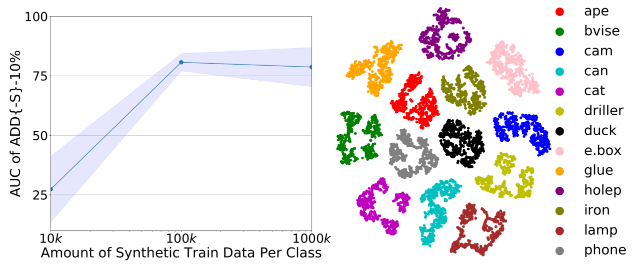
The results for using both real and synthetic data are summarized in Table III. As shown in Table III, using additional real training data boosts the performance of our system on all three datasets. On LM, ours with ICP has close performance to RGBD-based PVN3D. On LMO, among the methods without ICP, ours has the best performance. With ICP, PoseCNN (PC) outperforms our method by increasing their performance from 24.9% to 78.0%. One possible reason is that PoseCNN uses a sophisticated ICP while we use a standard point to point ICP. Moreover, it shows that LMO is very challenging for methods that rely on a single modality for pose inference, and combining both color and depth is beneficial. Another possible factor that is limiting our performance is that there might be a domain gap between our occlusion simulation and the real data. On YCBV, we report the results when using real training data with synthetic data from YCBV. We also combine the real training data from YCB and our synthetic data for training, and the testing accuracy is 93.6%. Using real and the two kinds of synthetic data achieves similar performance, and ours is comparable with the RGBD methods [12], [13].

More insights are gained by jointly comparing the results from Table II and Table III. On LMO, among the methods without ICP, ours has better performance compared to PVNet and PointVoteNet, which use real data for training. This shows our system can also handle occlusion. On YCBV, although trained on synthetic data only, our method has comparable performance to other methods trained on both real and synthetic data.

D. Ablation studies

We first study how the system performance is impacted by the amount of synthetic training data. For each object class in LM, we generate either $10k$, $100k$, $1000k$ training poses per class. For each pose data amount, we train multiple networks with early stopping. If the training error of the current epoch failed to improve more than 10% of the previous lowest training error, the training process is terminated. We conducted five trials for $10k$, and three trials for $100k$ and $1000k$. We generate a set of training data for each trial separately. The trained networks are evaluated on the test set of LM. Figure 4(a) shows the test accuracy (without ICP). $100k$ samples per class are sufficient for the network to perform well on the test set, and using $1000k$ per class does not provide further performance gain.

We also investigate how the performance is impacted by latent code size. We train networks with different sizes for the latent code. All the networks are trained with the same synthetic data and trained until convergence. The results without ICP are shown in Table IV. The generalization ability of a network is the best when the latent code is of size 1024 and 2048. We pick 1024 for our system. We show a T-SNE [35] visualization of the latent code for the LineMOD dataset in Figure 4(b). The latent codes are well clustered for each class.



(a) (b)

Fig. 4. Illustrations of ablation studies. (a) Amount of synthetic training data and performance accuracy (w/o ICP). (b) T-SNE visualization of latent code for the LineMOD dataset.

TABLE IV

RESULTS (W/O ICP) WITH DIFFERENT LATENT CODE SIZES ON LM.

Dimension	512	1024	2048	4096
avg	82.0	**82.1**	**82.1**	80.9

E. Runtime and hardware storage

We measure the time performance on an Nvidia Titan X GPU. Our system is implemented with Tensorflow. For the on-line data synthesis, it takes under 30 milliseconds to generate 128 object segments. The rendering based approach [36] takes 1-4 seconds to generate an image with 10-20 objects. Pose estimation by a forward pass through our network takes 0.07 seconds for a single object. The 10 iterations of ICP refinement require an additional 0.02 seconds. In the LM experiments, the 3D object models and $13 \times 100k$ training poses take $128\,\mathrm{MB}$ of storage. If to generate PBR images [2] with similar amount of objects (e.g. 10 objects per image), $13 \times 10k$ images would take $63\,\mathrm{GB}$.

VII. CONCLUSION

We propose a point cloud based lightweight data synthesis pipeline and an augmented autoencoder based system for accurate and fast 6D pose estimation of known objects represented by point clouds. The data synthesis pipeline is low cost in terms of both time and hardware storage. Using the synthetic data created by our data synthesis pipeline, our pose estimation system achieves state-of-the-art performance among other synthetic trained methods on a public benchmark. Moreover, our cheap synthetic point cloud data can replace expensive render based synthetic data for training systems using depth for pose inference. Our lightweight data synthesis pipeline enables more agile deployment of object pose estimation systems in robotic applications.

References

[1] A. Zeng, K. Yu, S. Song, D. Suo, E. Walker, A. Rodriguez, and J. Xiao, "Multi-view self-supervised deep learning for 6D pose estimation in the amazon picking challenge," in *ICRA*, 2017.

[2] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbe, E. Brachmann, F. Michel, C. Rother, J. Matas, and C. Rother, "BOP challenge 2020 on 6D object localization," in *ECCV Workshop on Recovering 6D Object Pose*, 2020.

[3] P. Marion, P. R. Florence, L. Manuelli, and R. Tedrake, "LabelFusion: A pipeline for generating ground truth labels for real RGBD data of cluttered scenes," in *ICRA*, 2018.

[4] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," in *RSS*, 2018.

[5] T. Hodaň, F. Michel, E. Brachmann, W. Kehl, A. Glent Buch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, and C. Rother, "BOP: Benchmark for 6D object pose estimation," in *ECCV*, 2018.

[6] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3D orientation learning for 6D object detection from RGB images," in *ECCV*, 2018.

[7] J. Wu, L. Ma, and X. Hu, "Delving deeper into convolutional neural networks for camera relocalization," in *ICRA*, 2017.

[8] G. Gao, M. Lauri, Y. Wang, X. Hu, J. Zhang, and S. Frintrop, "6D object pose regression via supervised learning on point clouds," in *ICRA*, 2020.

[9] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes," in *ACCV*, 2012.

[10] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D object detection from RGB-D data," in *CVPR*, 2018.

[11] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *CVPR*, 2017.

[12] C. Wang, D. Xu, Y. Zhu, R. Martín-Martín, C. Lu, L. Fei-Fei, and S. Savarese, "DenseFusion: 6D object pose estimation by iterative dense fusion," in *CVPR*, 2019.

[13] Y. He, W. Sun, H. Huang, J. Liu, H. Fan, and J. Sun, "PVN3D: A deep point-wise 3D keypoints voting network for 6DoF pose estimation," in *CVPR*, 2020.

[14] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*, vol. 11, no. 12, 2010.

[15] Y. Wen, H. Pan, L. Yang, and W. Wang, "Edge enhanced implicit orientation learning with geometric prior for 6D pose estimation," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4931–4938, 2020.

[16] S. Zakharov, I. Shugurov, and S. Ilic, "DPOD: 6D pose object detector and refiner," in *ICCV*, 2019.

[17] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6D object pose prediction," in *CVPR*, 2018.

[18] M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth," in *ICCV*, 2017.

[19] K. Park, T. Patten, and M. Vincze, "Pix2Pose: Pixel-wise coordinate regression of objects for 6D pose estimation," in *ICCV*, 2019.

[20] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield, "Deep object pose estimation for semantic robotic grasping of household objects," in *CoRL*, 2018.

[21] T. Hodaň, V. Vineet, R. Gal, E. Shalev, J. Hanzelka, T. Connell, P. Urbina, S. Sinha, and B. Guenter, "Photorealistic image synthesis for object instance detection," in *ICIP*, 2019.

[22] C. Mitash, K. E. Bekris, and A. Boularias, "A self-supervised learning system for object detection using physics simulation and multi-view pose estimation," in *IROS*, 2017.

[23] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views," in *ICCV*, 2015.

[24] F. Manhardt, W. Kehl, and A. Gaidon, "ROI-10D: monocular lifting of 2D detection to 6D pose and metric shape," in *CVPR*, 2019.

[25] S. Katz, A. Tal, and R. Basri, "Direct visibility of point sets," in *SIGGRAPH*, 2007.

[26] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Transactions on Graphics (TOG)*, 2019.

[27] G. Gao, M. Lauri, J. Zhang, and S. Frintrop, "Occlusion resistant object rotation regression from point cloud segments," in *ECCV 4th International Workshop on Recovering 6D Object Pose*, 2018.

[28] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[29] E. Brachmann, A. Krull, F. Michel, J. S. S. Gumhold, and C. Rother, "Learning 6D object pose estimation using 3D object coordinates," in *ECCV*, 2014.

[30] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again," in *ICCV*, 2017.

[31] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "PVNet: Pixel-wise voting network for 6dof pose estimation," in *CVPR*, 2019.

[32] F. Hagelskjær and A. Buch, "PointVoteNet: Accurate object detection and 6 DoF pose estimation in point clouds," in *ICIP*, 2020.

[33] B. Calli, A. Walsman, A. Singh, S. Srinivasa, and P. Abbeel, "Benchmarking in manipulation research using the Yale-CMU-Berkeley object and model set," *Robotics & Automation Magazine, IEEE*, vol. 22, no. 3, pp. 36–52, 2015.

[34] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[35] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[36] M. Denninger, M. Sundermeyer, D. Winkelbauer, Y. Zidan, D. Olefir, M. Elbadrawy, A. Lodhi, and H. Katam, "BlenderProc," *arXiv preprint arXiv:1911.01911*, 2019.