KeyPose: Multi-View 3D Labeling and Keypoint Estimation for Transparent Objects

Xingyu Liu[1*] Rico Jonschkowski[2] Anelia Angelova[2] Kurt Konolige[2]
[1]Stanford University [2]Robotics at Google

Abstract

Estimating the 3D pose of desktop objects is crucial for applications such as robotic manipulation. Many existing approaches to this problem require a depth map of the object for both training and prediction, which restricts them to opaque, lambertian objects that produce good returns in an RGBD sensor. In this paper we forgo using a depth sensor in favor of raw stereo input. We address two problems: first, we establish an easy method for capturing and labeling 3D keypoints on desktop objects with an RGB camera; and second, we develop a deep neural network, called KeyPose*, that learns to accurately predict object poses using 3D keypoints, from stereo input, and works even for transparent objects. To evaluate the performance of our method, we create a dataset of 15 clear objects in five classes, with 48K 3D-keypoint labeled images. We train both instance and category models, and show generalization to new textures, poses, and objects. KeyPose surpasses state-of-the-art performance in 3D pose estimation on this dataset by factors of 1.5 to 3.5, even in cases where the competing method is provided with ground-truth depth. Stereo input is essential for this performance as it improves results compared to using monocular input by a factor of 2. We will release a public version of the data capture and labeling pipeline, the transparent object database, and the KeyPose models and evaluation code. Project website:* https://sites.google.com/corp/view/keypose.

1. Introduction

Estimating the position and orientation of 3D objects is one of the core problems in computer vision applications that involve object-level perception such as augmented reality (AR) and robotic manipulation. Rigid objects with a known model can be described by 4D pose (e.g., vehicles [15, 12]), 6D pose [35, 4], and 9D pose where scale is predicted [33]. A more flexible method uses 3D *keypoints* [18, 30], which can handle articulated and deformable objects such as the human hand or body [29, 14]. While some of these methods predict 3D keypoints from a single RGB image, others use RGBD data collected by a depth sensor [32, 18, 2] to achieve better accuracy. Unfortunately, existing commercial depth sensors, such as projected light or time-of-flight (ToF) sensors, assume that objects have opaque, lambertian surfaces that can support diffuse reflection from the sensor. Depth sensing fails when these conditions do not hold, e.g., for *transparent* or *shiny metallic* objects. Figure 1 shows such an example.

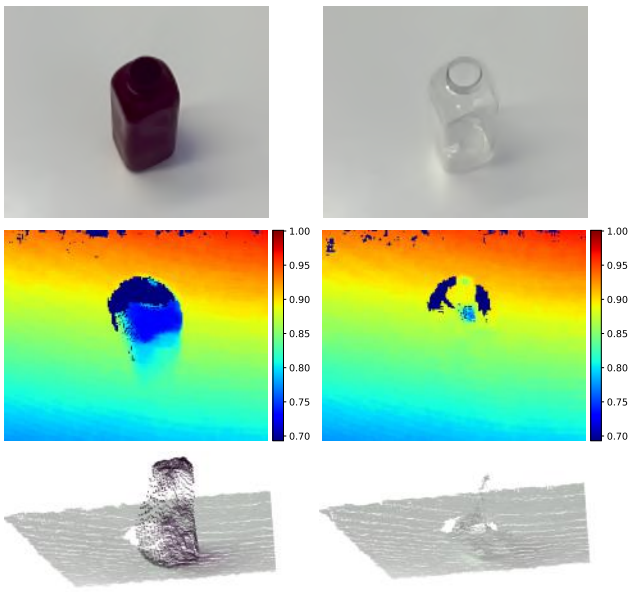

Figure 1: RGB image (top), depth map (middle), and point cloud (bottom) of an opaque bottle (left) and its transparent twin (right). The opaque bottle returns reasonable depth while the transparent one returns invalid depth values using a Microsoft Azure Kinect sensor.

In this paper, we present the first method of **keypoint-based pose estimation for (transparent) 3D objects from stereo RGB images**. There are several challenges: first, there is no available large-scale dataset for transparent 3D

*Work done as an intern at Google Research/Robotics at Google.

object pose estimation from stereo images with annotated keypoints. Datasets such as NYUDepth v2 [21] lack annotations for precise pose of each individual objects, while other datasets such as LabelFusion [18], YCB dataset [2] and REAL275 [33] annotate monocular RGBD images of opaque objects. The second challenge is the annotation of pose of transparent 3D objects. Existing datasets such as [18, 2, 33] require accurate depth information as well as an object CAD model so that alignment algorithms such as iterative closest point (ICP) [1] can be applied. The third challenge is how to leverage only RGB images for 3D keypoint estimation, thus obviating the need for a depth sensor.

To address the challenges regarding data acquisition and annotation, we introduce an efficient method of capturing and labeling stereo RGB images for transparent (and other) objects. Although our method does not need them, we also capture and register depth maps of the object, for both the transparent object and its opaque twin, registered with the stereo images; we use a robotic arm to help automate this process. The registered opaque depth allows us to compare to methods that require depth maps as input such as DenseFusion [32]. Following the proposed data capturing and labeling method, we constructed a large dataset consisting of 48k images from 15 transparent object instances. We call this dataset TOD (Transparent Object Dataset).

To reduce the requirement on reliable depth, we propose a deep model, KeyPose, that predicts 3D keypoints on transparent objects from cropped stereo RGB input. The crops are obtained from a detection stage that we assume can loosely bound objects (see [27] for an appropriate method for transparent objects). The model determines depth implicitly by combining information from the image pair, and predicting the 3D positions of keypoints for object instances and classes. After training on TOD, we compare KeyPose to the best existing RGB and RGBD methods and find that it vastly outperforms them on this dataset. In summary, we make the following contributions:

- A pipeline to label 3D keypoints on real-world objects, including transparent objects that does not require depth images, thus making learning-based 3D estimation of previously unknown objects possible without simulation data or accurate depth images. This pipeline supports a twin-opaque technique to enable comparison with models that require depth input.
- A dataset of 15 transparent objects in 6 classes, labeled with relevant 3D keypoints, and comprising 48k stereo and RGBD images with both transparent and opaque depth. This dataset can also be used in other transparent 3D object applications.
- A deep model, KeyPose, that predicts 3D keypoints on these objects with high accuracy using RGB stereo input only, and even outperforms methods which use ground-truth depth input.

2. Related Work

4D/6D/9D Pose Representation. The assumption behind these representations is the rigidity of the object, so that translation, rotation and size is sufficient to describe its configuration. Existing techniques for 4D/6D/9D pose estimation can generally be categorized by whether a 3D CAD model is used in training or inference. The first type of technique aligns the observed RGB images with rendered CAD model images [4, 13], or aligns the observed 3D point clouds with 3D CAD model point clouds with algorithms such as ICP [32], or renders mixed reality data from 3D CAD models as additional training data [33]. While it is possible to render high-quality RGB scenes of transparent objects using ray-tracing, there has been no work done on rendering depth images that faithfully reproduces the degraded depth seen in real-world RGBD data (see Figure 1).

The second type of technique regresses the object coordinate values from the RGB image or 3D point clouds [35, 15, 12, 24, 25]. Our method does not assume object rigidity, and the object pose is based the locations of 3D keypoints, which can be used on articulated or deformable objects. Our method also does not rely on prior knowledge about each individual object, such as a 3D CAD model.

Keypoint Based Pose Representation. Previous work has explored deep learning methods for detecting 3D keypoints of an object given a monocular RGB image [30] or RGBD image [17]. The core is to predict probability maps for the 2D keypoint locations, and then use the given or predicted depth image for 3D. Other works proposed similar methods for monocular pose estimation [29, 20, 31]. Though estimating 3D positions from a single RGB image is an ill-conditioned problem, these methods implicitly learn the prior of object size during training, or rely on the known object 3D model. Our method is inspired by these works and focuses on 3D keypoint location estimation from stereo instead of single images, and is well-conditioned even for similar objects that vary in scale. Recently, a method similar to ours was proposed for hand tracking using raw strereo [14]. For rigid objects with a known model, the 6D pose can be recovered using the Procrustes algorithm (see the Supplementary materials).

Stereo for Disparity Estimation. Estimating disparity and therefore depth from stereo has been a long-standing problem in computer vision. The success of deep-learning methods in computer vision inspired research in this area, using end-to-end deep networks equipped with a correlation cost volume [19, 11, 5, 36], or point-based depth representation and iterative refinement [3]. Here, instead of generating a dense disparity field, we focus on estimating the 3D location of sparse keypoints directly from stereo images.

3D Object Pose Estimation Datasets. Directly labeling 3D object pose in real RGB images is costly. All existing real (non-synthetic) datasets for 3D object pose estima-

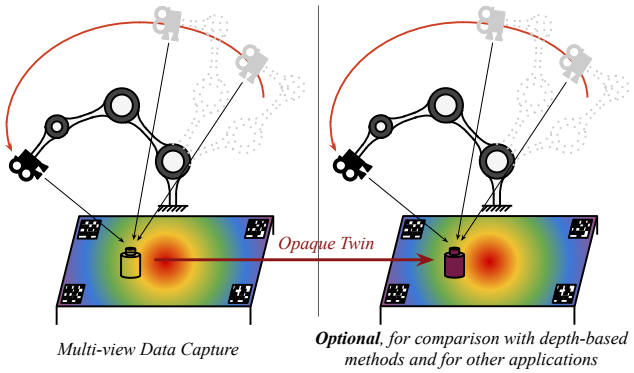

Figure 2: Data capturing pipeline. We mount both the stereo RGB camera and RGBD camera on the end-effector of the robot. We then use the robot arm to perform similar paths to scan both the opaque lambertian object (left) and its transparent twin placed at the same location of a textured surface (right). AprilTags [34] are used as global pose indicator for the cameras.

tion rely on capturing RGBD images and annotating pose by either constructing a 3D mesh [17], or fitting 3D CAD models to 3D point clouds [18, 2, 33, 9], neither of which is possible for transparent objects. On the contrary, we build a data capturing pipeline where ground-truth depth of transparent object keypoints can be efficiently obtained, without relying on depth or 3D CAD models.

Estimation of transparent and reflective objects. Objects that are transparent or reflective present significant challenges for all camera-based depth estimation. Works on estimating transparent object pose and geometry might assume knowing object 3D model [23, 16] or rely on synthetic data to train vision models [28, 27]. Our data capturing and labeling enables generating large-scale real dataset for training and testing transparent object pose and geometry, so synthetic data are not needed.

3. Transparent Object Dataset (TOD)

In this section, we describe the data capturing pipeline that enables efficient capture and labeling of 3D keypoints for a large number of samples without requiring a depth sensor.

3.1. Data Collection with a Robot

Hand-labeling 3D keypoints in individual RGB images is difficult or impossible due to uncertainty about keypoint depth. Instead, we leverage multi-view geometry to raise 2D labels from a small number of images into 3D labels for a set of images where the object has not moved. The general idea is illustrated in Figure 2.

We use a stereo camera with known parameters to capture images in a sequence, moving the camera with a robot arm (we could also move it by hand). To estimate the pose of the camera relative to the world, we set up a planar form with AprilTags [34] that can be be recognized in an image, and from their known locations estimate the camera pose. From a small subset of widely-separated poses, we label 2D keypoints on the object. Optimization from multi-view geometry gives the 3D position of the keypoints, which can be reprojected to all images in the sequence. To increase diversity, we place various textures under the object. Figure 3 shows some challenging data examples.

Figure 3: Challenging cases in our dataset, including dark background textures (left), thin handles of mugs (middle) and motion blur (right). Accurately locating these objects is a difficult task even for human.

The resultant labeled stereo samples are sufficient to train and evaluate the KeyPose model. We can collect and label data for a new object in a few hours. In addition to the stereo data, we also capture and register depth data using the Microsoft Kinect Azure RGBD device. This data is purely ancillary to our model, but it lets us compare KeyPose to methods that require depth data. We collect two depth images, one during the initial scan with co-mounted stereo and RGBD devices, and one with the transparent object replaced by its opaque (painted) twin during a second scan (Figure 2, right). Although the RGBD images are captured at slightly different poses from the stereo (due to variations in the trajectories and camera capture times), we can leverage the calculated pose of the RGBD camera (using AprilTags in the RGB image), and the known offset of the depth sensor from the RGB sensor, to warp the depth image to align precisely with the left stereo image (see Figure 1).

3.2. Keypoint Labeling and Automatic Propagation

To accurately construct this dataset, we need to address different sources of error. First, since AprilTag detection is imperfect in finding tag positions, we spread out these tags on the target to produce large baselines for camera pose estimation. Second, since human labeling of keypoints on 2D images introduces error, we use a farthest-point algorithm on the camera poses to ensure that annotated images used in going from 2D to 3D have a large baseline.

We want to know the accuracy of the manual annotation. While the absolute ground truth of the 3D keypoints is unknown, we can estimate the labeling error, given the known reprojection errors of the AprilTags and 2D annotations. Using a Monte Carlo simulation based on the repro-

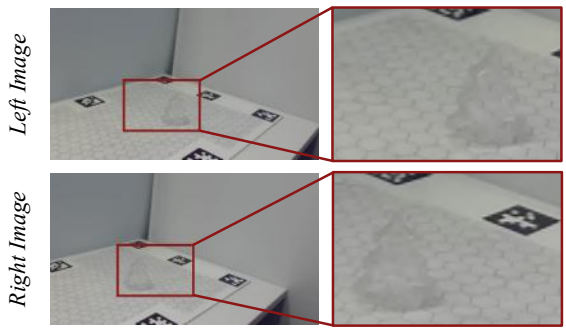

Figure 4: Example of cropping with bounding box for left and right images.

jection errors, we calculate the random error of the labeled 3D keypoints to be around 3.4 mm RMSE, which is quite accurate. Details of the simulation are in the supplementary material.

4. Predicting 3D Keypoints from RGB Stereo

In this section, we describe the KeyPose method of estimating the pose of 3D objects from stereo input, using supervised training of 3D keypoints. We first introduce patching cropping from bounding box and then describe our CNN architecture. Finally, we present the choice of loss functions used in training, which significantly affect the performance.

4.1. Data Input to the Training Process

We assume a detection stage that approximately determines the location of an object (see [27] for a method to detect and segment transparent objects; or, the *UV* heatmap of Figure 5 could be used). From this bounding box we crop a fixed-size rectangle from the left image, and a corresponding rectangle at the same height from the right image, preserving the epipolar geometry (Figure 4).

Since the right object image is offset from the left – in our case, by 48 to 96 pixels, given the stereo device and assuming an object distance from 0.5m to 1m – the rectangle must extend far enough to encompass the right object no matter where it might appear. To limit the rectangle extension, we offset the right crop horizontally by 30 pixels, changing the apparent disparity to 18-66 pixels. The input size for each crop is 180×120 pixels.

The input images are processed by the model to produce, for each keypoint, a UV (2D) image location of the keypoint and a disparity D that encodes depth and is the offset of the left and right keypoints (in pixels). The *UVD* triplet encodes the 3D *XYZ* coordinates by: $Q := UVD \mapsto XYZ$, where Q is a *reprojection matrix* determined by the camera parameters [22]. We use these *XYZ* positions as labels to generate training errors, by projecting back to the camera image and comparing *UVD* differences. Reprojected pixel errors are a stable, physically-realizable error method widely used in multiple-view geometry [10]. Comparing 3D errors directly introduces a large bias, as they grow quadratically with distance, overwhelming the errors of closer objects.

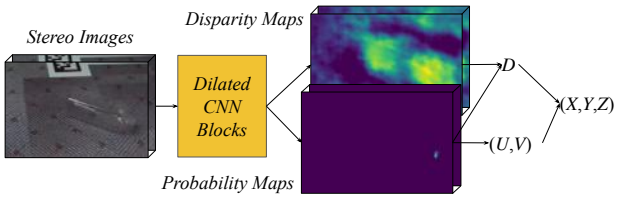

Figure 5: Early fusion architecture.

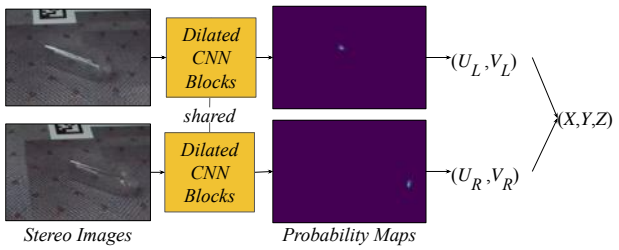

Figure 6: Late fusion architecture.

To encourage generalization, we perform geometric and photometric augmentation of the input images. More details are in the supplementary material. Note that geometric augmentation must be limited to transformations that do not violate epipolar constraints, i.e. scaling, Y-axis shear, mirroring, and rotation of the view around the X-axis.

4.2. Architecture for 3D Pose Estimation

The KeyPose model combines the following principles:

Stereo for Implicit Depth. Use stereo images to introduce depth information to the model.

Early Fusion. Combine information from the two image crops as early as possible. Let the deep neural network determine disparity implicitly, rather than forming explicit correlations (as in [5]).

Broad Context. Extend the spatial context of each keypoint as broadly as possible, to take advantage of any relevant shape information of the object.

Figure 5 shows the basic structure of the model, which was adapted from [30]. Stereo images are stacked and fed into a set of exponentially-dilated 3x3 convolutions [37] that expands the context for predicting keypoints, while keeping the resolution constant. Two such groupings ensure that the context for each keypoint is thoroughly mixed. The number of features is kept constant at 48 (for instance models) and 64 (for category models) throughout the CNN blocks. After this, *projection heads*, one per keypoint, extract *UVD* coordinates. We investigate two projection methods:

1. Direct regression. Three 1x1 convolutional layers produce $N \times 3$ numeric *UVD* coordinates, where N is the

number of keypoints.

2. Heatmaps. For each keypoint i, a CNN layer produces a heatmap, followed by spatial softmax to generate a probability map $prob_i$, and then integrated to get *UV* coordinates, as in IntegralNet [29]. A disparity heatmap is also computed, convolved with the probability map, and integrated to produce disparity (Figure 5). This method is also useful for visualization.

To test the efficacy of early fusion, we also implemented a *late fusion* model (Figure 6), in which siamese dilated CNN blocks separately predict *UV* keypoints for the left and right images. Then standard stereo geometry is used to generate a 3D keypoint prediction.

4.3. Losses

We use three losses: a direct keypoint *UVD* loss, a projection loss, and a locality loss. We also permute the total loss and take a minimum for symmetric keypoints.

Keypoint loss. The predicted (UVD) and labeled (UVD^*) pixel values are compared via squared error

$$\mathcal{L}_{kp} = \sum_{i \in kps} \|UVD_i - UVD_i^*\|^2 \tag{1}$$

We tried a direct 3D loss, but the errors grow quadratically with distance, overwhelming the errors of closer objects. This introduces a large bias in the model performance.

Projection loss. Predicted *UVD* values are converted to a 3D point, then re-projected to the widely-separated views that were used to create the 3D points. The difference between the predicted and labeled *UV* re-projections is squared for the loss. Let P_j be the projection function, and $Q := UVD \mapsto XYZ$. Then

$$\mathcal{L}_{proj} = \sum_{i \in keypts} \sum_{j \in views} \|P_j Q(UVD_i) - P_j Q(UVD_i^*)\|^2 \tag{2}$$

In the same way that the wide viewpoints pinpoint the 3D coordinates of a keypoint in generating labels, here they recreate the same conditions for constraining the predicted keypoint. This loss is critical for good performance ([10], and see Section 5.3).

Locality loss. Although the keypoint location is estimated from the *UV* probability map, that map might not be unimodal and might have high probabilities away from the true keypoint location. This loss encourages the probability map to localize around the keypoint.

$$\mathcal{L}_{loc} = \sum_{i \in kps} \sum_{uv} prob_i[uv] \cdot \widetilde{\mathcal{N}}(UV_i^*, \sigma)[uv] \tag{3}$$

$\mathcal{N}$ is a circular normal distribution centered on the labeled UV_i^* coordinates for keypoint i, with standard deviation σ.

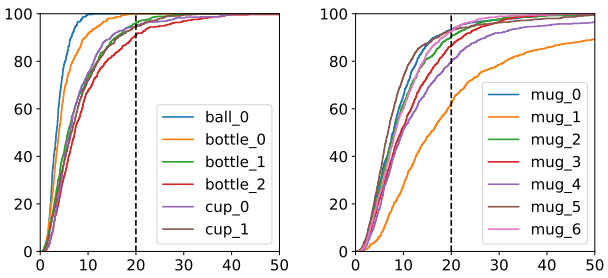

Figure 7: Precision curves for object instances. Y axis is cumulative percent. X axis is 3D MAE in mm; note it is limited to 50 mm instead of the normal 100 mm, to magnify the curves.

$\widetilde{\mathcal{N}}$ is a normalized inverse

$$1 - \mathcal{N} / \max(\mathcal{N}). \tag{4}$$

This loss gives a very low value when the predicted UV probability is concentrated near the UV label. We use a σ of 10 pixels.

The total loss is defined as the weighted sum

$$\mathcal{L}_{total} = \mathcal{L}_{kp} + \alpha \mathcal{L}_{proj} + 0.001 \mathcal{L}_{loc} \tag{5}$$

A small weight on $\mathcal{L}_{loc}$ nudges the probability distribution into the correct local form, while allowing room to spread out if necessary. For stability, it is important to apply a curriculum to $\mathcal{L}_{proj}$. The weight α ramps from 0 to 2.5 over the interval $[1/3, 2/3]$ of the training steps, to allow the predicted *UVD* values to stabilize. Otherwise, convergence can be difficult because the re-projection error gradients might initially be very large.

Permutation for symmetric keypoints. Symmetric objects can cause aliasing among keypoints ids. For example, the tree object in Figure 9 is indistinguishable when rotated 180° around its vertical axis. A keypoint placed on the object may thus obtain different, indistinguishable positions from the point of view of the pose estimator.

We deal with keypoint symmetry by allowing permutation of the relevant keypoint ids in the loss function. For example, in the case of the tree, there are two allowed permutations of the keypoint ids, $[1, 2, 3, 4]$ and $[1, 2, 4, 3]$. $\mathcal{L}_{total}$ is evaluated for each of these permutations, and the minimum is chosen as the final loss.

Keypoints handle some symmetries without any permutations. These are illustrated by the ball, bottle, and cup objects. For the ball, a single keypoint at the center confers full rotational symmetry. For bottles and cups, two keypoints along the cylindrical axis confer cylindrical symmetry. Note that we may choose to use fewer keypoints than necessary – for example, if we do not care where the handle of a mug is, we could use only the top and bottom keypoints.

method	input modality	metrics	ball	bottle_0	bottle_1	bottle_2	cup_0	cup_1	mug_0	mug_1	mug_2	mug_3	mug_4	mug_5	mug_6	heart	tree	mean
DenseFusion [32]	mono RGB + **opaque** depth	AUC↑	90.0	88.6	69.1	56.0	84.0	80.7	67.8	66.3	71.4	70.0	69.0	76.8	51.2	61.7	75.5	71.9
		<2cm↑	94.4	97.8	9.1	28.4	79.1	65.3	12.5	10.3	28.1	20.3	4.7	41.9	3.1	17.2	50.9	37.5
		MAE↓	9.9	11.3	57.6	77.8	16.0	37.5	32.2	33.7	28.6	30.0	31.0	23.2	75.2	38.3	24.5	35.1
DenseFusion [32]	mono RGB + **real** depth	AUC↑	84.7	81.6	72.3	47.5	59.4	77.8	54.5	51.3	60.4	67.3	48.1	70.6	64.9	61.2	55.6	63.8
		<2cm↑	78.8	67.5	18.1	9.1	5.6	54.4	4.6	0.3	12.2	8.1	0.0	20.0	4.7	0.0	0.0	18.9
		MAE↓	15.3	18.4	27.6	65.6	40.5	22.1	45.5	48.7	39.5	32.7	54.9	29.4	35.9	38.8	44.4	37.2
Ours	stereo RGB	AUC↑	**96.1**	**95.4**	**94.9**	**90.7**	**93.1**	**92.0**	**91.0**	**78.1**	**89.7**	**88.6**	**87.8**	**91.0**	**90.3**	**84.3**	**87.1**	**90.0**
		<2cm↑	**100**	**99.8**	**99.7**	**91.4**	**97.8**	**95.3**	**94.6**	**63.6**	**90.1**	**87.2**	**87.1**	**93.1**	**92.2**	**77.2**	**82.5**	**90.1**
		MAE↓	**3.8**	**4.6**	**5.1**	**9.3**	**6.8**	**7.1**	**8.9**	**21.9**	**10.1**	**11.3**	**12.1**	**9.0**	**9.7**	**15.6**	**12.8**	**9.9**

Table 1: Instance-level pose estimation results. For each object instance, the model is trained on nine background textures and evaluated on unseen textures. Higher is better for AUC and $<$ 2cm, lower for MAE.

method	DenseFusion [32]			DenseFusion [32]			**Ours**		
input modality	monocular RGBD + **opaque** depth			monocular RGBD + **real** depth			stereo RGB		
metrics	AUC↑	<2cm↑	MAE↓	AUC↑	<2cm↑	MAE↓	AUC↑	<2cm↑	MAE↓
bottles	83.4	88.4	34.2	76.9	71.0	26.4	**94.2**	**97.8**	**5.8**
bots+cups	90.0	93.4	10.5	77.2	70.3	24.5	**93.4**	**97.8**	**6.6**
mugs	82.4	72.8	17.6	73.5	41.5	26.5	**90.1**	**92.6**	**9.9**

Table 2: Category-level pose estimation results. Evaluate on unseen textures. Boldface are best results.

4.4. Training

We trained the KeyPose model with a batch size of 32 and a constant number of steps, around 300 epochs. For DenseFusion, we re-implemented the algorithm in TensorFlow and trained until convergence, around 80 epochs. Since DenseFusion does not return keypoints, we added layers to regress to 3D positions for each keypoint. More training details are in the supplementary material.

5. Experiments

We conducted experiments to test the KeyPose model and DenseFusion [32], on the TOD dataset. We compared two input variants for DenseFusion, with depth from opaque and transparent (real) versions of the object. Remember that in the case of opaque depth, we still use the RGB image of the transparent object. We trained both instance and category models, and derived test sets by holding out all sequences of a texture, and all sequences of an object. We also performed ablation studies to understand the effects of stereo and the various losses.

Two error measures that are standard in the literature [4, 32, 35] are Area Under the Curve (AUC) and percentage of 3D keypoint errors $<$2cm. AUC percentage is calculated based on an X-axis range from 0 to 10 cm, where the curve shows the cumulative percentage of errors under that metric value (Figure 7). These measures were developed for lower-accuracy methods, and we prefer a more precise measure, Mean Absolute Error (MAE) of the 3D keypoints.

method	DenseFusion [32]			DenseFusion [32]			**Ours**		
input modality	monocular RGBD + **opaque** depth			monocular RGBD + **real** depth			stereo RGB		
metrics	AUC↑	<2cm↑	MAE↓	AUC↑	<2cm↑	MAE↓	AUC↑	<2cm↑	MAE↓
mugs	76.4	40.7	23.5	74.3	43.4	25.7	**84.7**	**78.6**	**15.6**

Table 3: Pose estimation for the mug category. Evaluate on unseen instance mug_0.

5.1. Instance-Level Pose Estimation

Each of the 15 objects was trained separately, and statistics computed for a held-out texture. There were approximately 3000 training samples and 320 test samples. This experiment captures how well an instance-level model can generalize in a new setting. The results are illustrated in Table 1. Not surprisingly, DenseFusion(opaque) performed better than DenseFusion(real) in almost every case, with the exception of cup_1 and mug_6. These latter may be due to the errors in depth from the depth device, which even in the opaque case can have significant errors – see the Supplemental. For both, the 3D errors were large, averaging over 35 mm across the dataset.

KeyPose out-performed DenseFusion(real) across-the-board, often by large amounts. Surprisingly, it also performed better than DenseFusion(opaque) on all objects. This is despite the large premium offered by good depth information for the latter. KeyPose MAE was 9.9 mm, averaged over all objects, more than 3.5 times more accurate than DenseFusion. These results demonstrate that KeyPose with stereo input works remarkably well for transparent objects. Given its performance relative to DenseFusion(opaque), it is capable of surpassing state-of-the-art results for pose estimation of desktop object instances.

5.2. Category-Level Pose Estimation

We defined three categories: bottles (3 objects), bottles and cups (5 objects), and mugs (7 objects). For each category, we trained DenseFusion and KeyPose models, leaving out one texture over all objects as the test set. Thus, this experiment captures how well a category-level model can generalize to any of its members in a new setting. From the

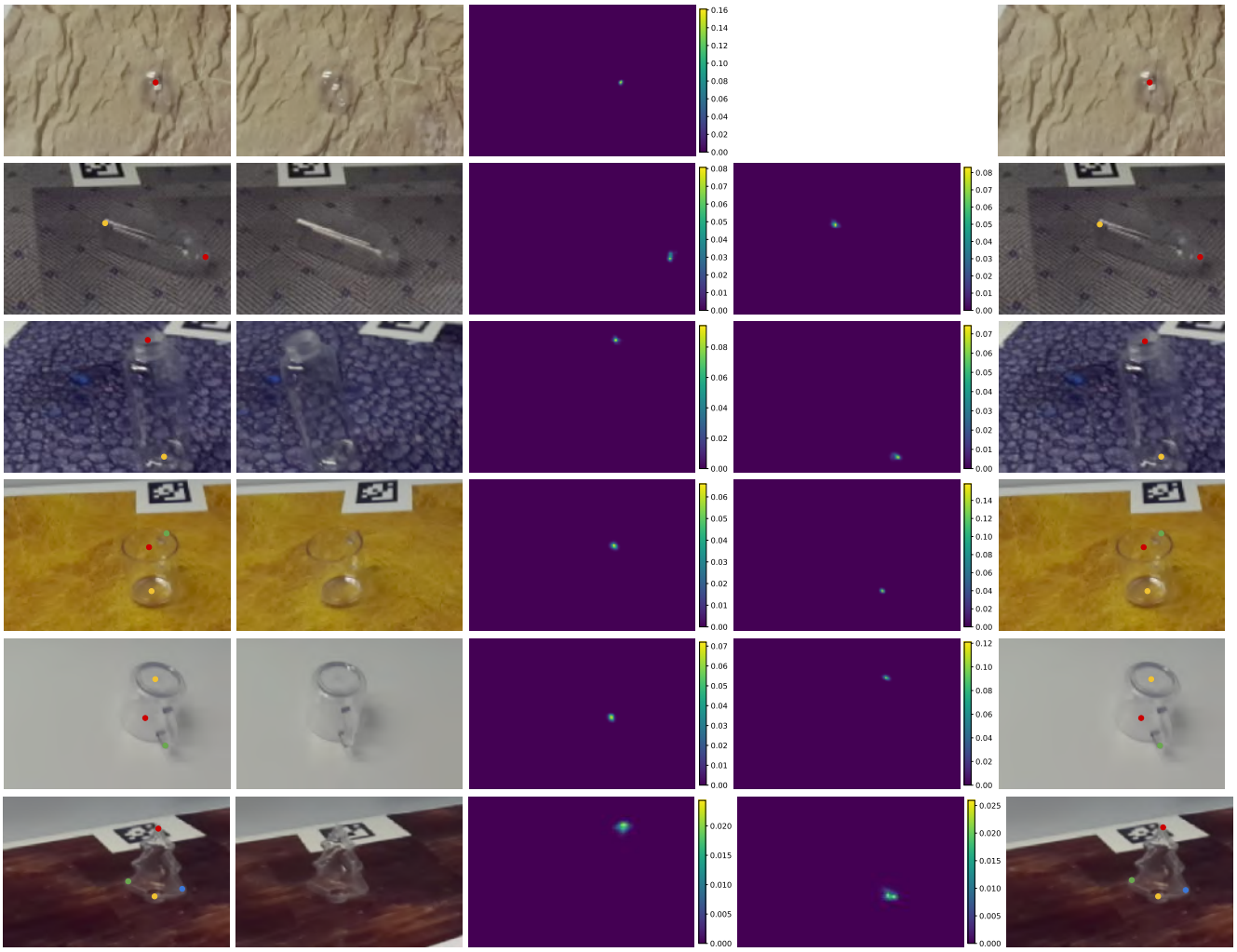

Figure 8: Visualization of prediction results on validation set. From left to right in each row: left stereo image with groundtruth keypoints, right stereo image, predicted probability map for first keypoint, predicted probability map for second keypoint, and predicted keypoints. We use red, yellow, green and blue to mark keypoint 1, 2, 3 and 4.

results in Table 2, KeyPose surpasses DenseFusion in accuracy by factors of 2 to 5. Both methods seem to benefit from having larger numbers of samples for training.

In a second category-level experiment, we held out mug_0 for testing; this experiment shows how well the methods generalize to unseen objects. Given the small number of mugs in the category, the result shows surprisingly good generalization (Table 3). KeyPose is more accurate than DenseFusion(opaque and real), by a factor of 1.5. With more objects in the mug category, it is likely both methods would improve.

5.3. Ablation Studies

To find out which parts of KeyPose are effective, we performed an ablation study on the losses and architecture (stereo vs. monocular, early fusion vs. late fusion, regression vs. integration, projection loss, permutation loss), for a selection of instance and category models. Results are in Tables 6, 4.

First, note that **using stereo improves accuracy over monocular input by a factor of 2**, for both instance and category training. Although monocular systems can gauge distance by the apparent size of an item, oblique views and different-size objects can make this difficult. The most telling difference is in the disparity error, where it grows to almost a pixel, while stereo is at half that. This clearly demonstrates that stereo input is being used by the network to determine distance, and that keeping the disparity error low is the key to good 3D estimation.

There is similar but smaller difference between early and late fusion. Recall that in late fusion (column 2), keypoints are computed for both left and right images, and then their

stereo		✗	✓	✓	✓	✓
early fusion		✓	✗	✓	✓	✓
projection loss		✓	✓	✗	✓	✓
direct regression		✓	✓	✓	✗	✓
bottle_0	3D MAE (mm)	10.0	7.9	5.4	4.7	**4.6**
	UV MAE (px)	1.62	**1.07**	1.08	1.14	1.21
	Disp MAE (px)	0.91	0.67	0.45	0.38	**0.36**
bottles	3D MAE (mm)	10.1	10.6	9.9	6.0	**5.8**
	UV MAE (px)	1.23	1.38	**1.30**	1.41	1.37
	Disp MAE (px)	0.94	0.96	0.89	**0.47**	0.48

Table 4: Ablation study for architecture and loss functions.

crop size	180x120	270x180	360x240
3D MAE (mm)	**4.6**	5.0	5.3

Table 5: Ablation study on size of crop region for bottle_0.

U-values are compared to give the disparity. Since the U-values have low error, the disparity values do, too. However, they are higher than in early fusion, which can take advantage of mixing information from both images in the network. We also observed a much longer error tail for late fusion, with some large metric errors.

The projection loss $\mathcal{L}_{proj}$ (column 3) helps to keep the disparity error low. Without it, disparity errors are higher, having 0.09 pixels more in the instance case, and 0.41 pixels more in the category case. The *UV* error is actually lower when not using the projection loss, but it is less important. While 0.41 pixels may not seem like a large difference, it can have an outsize effect on metric error. From stereo geometry, the change of depth for a change of disparity is given by:

$$\frac{\Delta z}{\Delta d} = -\frac{z^2}{fb} \tag{6}$$

where f is the focal length and b is the baseline. At an object distance of 0.8 m, for example, 0.41 pixel error in disparity yields a 5.5 mm error in depth for our stereo system.

The difference between using direct regression to *UVD* values, vs. an integral approach, shows a small bias in favor of regression. The advantage of the integral approach is the production of *UV* and disparity maps, which is useful for visualization of the network predictions (see Figures 5, 8).

For the permutation loss, results are in Table 6. We examined the tree object and turned off the permutation of the side keypoints. Figure 9 shows the effect: since the sides of the tree are symmetric, the choice of which keypoint will be labeled is random. Training without the permutation loss causes the two keypoints to cluster in the center to minimize loss. This is reflected in the wide difference between the results.

Finally, we consider whether the results depend on a tight crop of the object. First note that the crop is generous, especially for small objects (see Figure 8). Then, we

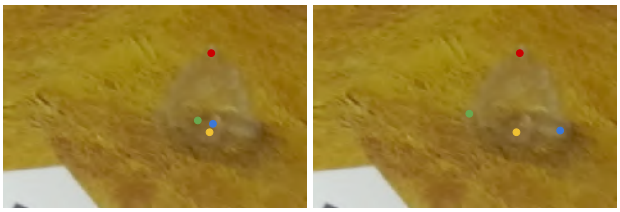

Figure 9: Visualization of ablation study: without (left) vs. with (right) permutation loss. We use red, yellow, green and blue to mark keypoint 1, 2, 3 and 4. Instance tree_0 has symmetric keypoints 3 & 4.

metrics		3D MAE (mm)	UV MAE (px)	Disp MAE (px)
perm	✗	26.4	11.1	1.46
loss	✓	**12.8**	**2.79**	**1.05**

Table 6: Ablation study on tree_0 for permutation loss.

dither the location of the object by 20 pixels to make KeyPose robust to bounding box placement. We also checked larger crops in Table 5, up to 4 times the original area. The results show minimal degradation, less than any of the loss ablations in Table 4. Many CNN methods use a tight crop of an object followed by scaling to present the same size to the network. Here we have chosen the harder problem and used a fixed size crop with no rescaling. The apparent size of objects varies by a factor of about 2.5, which is reasonable for a lot of applications, such as bin-picking with fixed cameras. It remains to future work to see if tight crops and scaling would be more accurate.

6. Conclusion and Discussion

In this paper, we studied the problem of estimating the 3D object pose represented by 3D keypoint locations from stereo images. By providing an easy-to-use 3D keypoint labeling facility, we have generated TOD, a large-scale labeled dataset of transparent objects, along with registered depth, for training and comparing keypoint pose estimation methods. The KeyPose model, utilizing early fusion of stereo images, surpasses state-of-the-art on all benchmark tests in instance and category levels, including when opaque depth is used. It generalizes across unseen textures and to unseen objects. The ablation studies validate our emphasis on early fusion and multi-view reprojection losses.

There are some areas that need further improvement and exploration. Among these are detecting transparent objects using our heatmap technique, adding more complex backgrounds, varying lighting, and including multi-object samples into the dataset. We will also investigate using mobile robots to capture data in the wild. Although we concentrated on transparent rigid objects, KeyPose can also be applied to opaque, articulated and deformable objects. These directions will be left as future work.

References

[1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE TPAMI*, 1992. 2

[2] Berk Calli, Aaron Walsman, Arjun Singh, Siddhartha Srinivasa, Pieter Abbeel, and Aaron M. Dollar. Benchmarking in manipulation research: The ycb object and model set and benchmarking protocols. *CoRR*, abs/1502.03143, 2015. 1, 2, 3

[3] Rui Chen, Songfang Han, Jing Xu, and Hao Su. Point-based multi-view stereo network. In *ICCV*, 2019. 2

[4] Xinke Deng, Arsalan Mousavian, Yu Xiang, Fei Xia, Timothy Bretl, and Dieter Fox. Poserbpf: A rao-blackwellized particle filter for 6d object pose tracking. In *RSS*, 2019. 1, 2, 6

[5] Xianzhi Du, Mostafa El-Khamy, and Jungwon Lee. Amnet: Deep atrous multiscale stereo disparity estimation networks. *arXiv preprint arXiv:1904.09099*, 2019. 2, 4

[6] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 1997. 16

[7] Teofilo F Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985. 16

[8] John C Gower. Generalized procrustes analysis. *Psychometrika*, 1975. 18

[9] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *ICCV*, 2011. 3

[10] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *CVPR*, 2017. 4, 5

[11] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *ICCV*, 2017. 2

[12] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 1, 2

[13] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. Deepim: Deep iterative matching for 6d pose estimation. In *ECCV*, 2018. 2

[14] Yuncheng Li, Zehao Xue, Yingying Wang, and Zhou Ren. End-to-end 3 d hand pose estimation from stereo cameras. In *BMVC*, 2019. 1, 2

[15] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *CVPR*, 2018. 1, 2

[16] Ilya Lysenkov and Vincent Rabaud. Pose estimation of rigid transparent objects in transparent clutter. In *ICRA*, 2013. 3

[17] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019. 2, 3

[18] Pat Marion, Peter R. Florence, Lucas Manuelli, and Russ Tedrake. Labelfusion: A pipeline for generating ground truth labels for real rgbd data of cluttered scenes. In *ICRA*, 2018. 1, 2, 3

[19] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 2

[20] Dushyant Mehta, Helge Rhodin, Dan Casas, Pascal Fua, Oleksandr Sotnychenko, Weipeng Xu, and Christian Theobalt. Monocular 3d human pose estimation in the wild using improved cnn supervision. In *3DV*, 2017. 2

[21] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 2

[22] OpenCV. reprojectImageTo3D function. `https://docs.opencv.org/2.4`, 2020. 4

[23] Cody J Phillips, Matthieu Lecce, and Kostas Daniilidis. Seeing glassware: from edge detection to pose estimation and shape recovery. In *RSS*, 2016. 3

[24] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *CVPR*, 2018. 2

[25] Mahdi Rad and Vincent Lepetit. BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *CoRR*, 2017. 2

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. 17

[27] Shreeyak S Sajjan, Matthew Moore, Mike Pan, Ganesh Nagaraja, Johnny Lee, Andy Zeng, and Shuran Song. Clear-grasp: 3d shape estimation of transparent objects for manipulation. *arXiv preprint arXiv:1910.02550*, 2019. 2, 3, 4

[28] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *IJRR*, 2008. 3

[29] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral human pose regression. In *ECCV*, 2018. 1, 2, 5, 17

[30] Supasorn Suwajanakorn, Noah Snavely, Jonathan Tompson, and Mohammad Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *NIPS*, 2018. 1, 2, 4, 16

[31] Bugra Tekin, Federica Bogo, and Marc Pollefeys. H+O: unified egocentric recognition of 3d hand-object poses and interactions. *CoRR*, 2019. 2

[32] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6d object pose estimation by iterative dense fusion. In *CVPR*, 2019. 1, 2, 6, 18

[33] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J. Guibas. Normalized object coordinate space for category-level 6d object pose and size estimation. In *CVPR*, 2019. 1, 2, 3

[34] John Wang and Edwin Olson. Apriltag 2: Efficient and robust fiducial detection. In *IROS*, 2016. 3

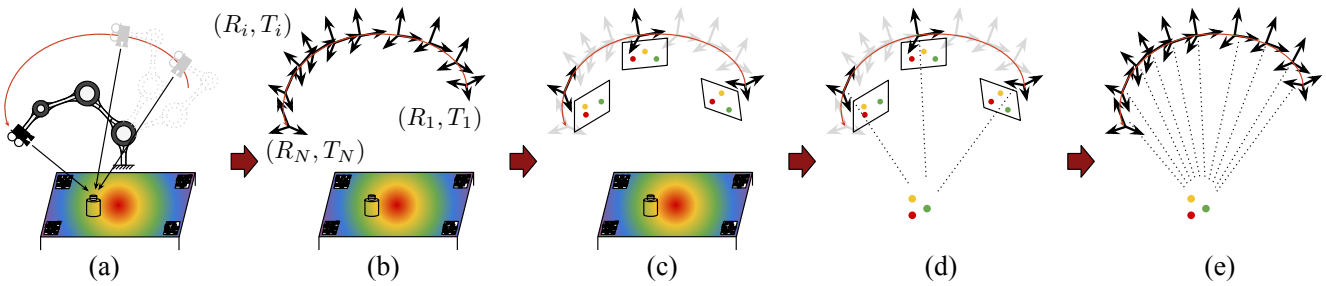

Figure 10: Labeling pipeline. (a) We use robot to scan the object from different views and record stereo-RGB and RGBD sequences; (b) AprilTags groundtruth locations are used to calculate the global pose of each frame (1 through N) in the video based on Perspective-n-Point (PnP) algorithm; (c) Only a few key frames selected from the video sequence are labeled, where the selection is based farthest point sampling (FPS) of camera poses; (d) From the labeled 2D locations of the keypoints, the 3D locations of the keypoints are calculated; (e) The 3D locations are propagated to all frames in the sequence to obtain the 2D projected UV location and depth.

[35] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. In *RSS*, 2018. 1, 2, 6

[36] Guorun Yang, Hengshuang Zhao, Jianping Shi, Zhidong Deng, and Jiaya Jia. Segstereo: Exploiting semantic information for disparity estimation. In *ECCV*, 2018. 2

[37] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. 4, 17

Supplementary

A. Overview

In this document, we provide additional detail on KeyPose as presented in the main paper. We present object examples in Section B. We present details of data capturing pipeline and error analysis in Section C. We also provide details of the architectures, training procedure, and timing in Section D.

B. Object and Texture Examples

Placement of Object Twins. Capturing the groundtruth depth of the transparent object requires placing its opaque twin at the same pose. We proposed an efficient way to accurately do so. The process is illustrated in Figure 11, where we show the replacement of mug_3.

We first place the transparent object at a desired pose and scan the RGBD and stereo RGB video. Then we align a specially designed plastic marker closely to the object. The plastic marker consists of three sticks orthogonal to each other so that the marker's relative configuration with respect to the object is unique. After alignment, we remove the transparent object but keep the marker from moving. Next, we place the opaque twin so that it closely aligns with the marker in the same configuration as the transparent object. Finally, we remove the marker but keep the opaque twin from moving. In this case, the transparent object and the opaque twin will have exactly the same pose.

Objects Used in Our Dataset. Our complete dataset consists of 20 object pairs in total, though only 15 object pairs are used in the experiments of the main paper. We illustrate them and the keypoint groundtruth definition in the first column of Figure 13, 14, 15 and 16. We also scan the opaque objects and provide the 3D CAD model of the objects for applications that require them. The CAD models of the objects and the alignment of the markers to the objects are also illustrated in Figure 13, 14, 15 and 16.

Textures Used in Our Dataset. In our dataset, each object in placed on ten diverse background textures, which are illustrated in Figure 17. We print the textures on papers and place them beneath the objects. The textures include pebbles, rocks, woods, textile etc.

C. Data Capture and Error Analysis

In this section we provide more detail about the data capture pipeline, as well as an analysis of pose estimation and 3D keypoint errors. The pipeline for capturing and labeling a single object is illustrated in Figure 10.

C.1. Data Capture Pipeline

Data is captured via a sensor head attached to a Franka Panda arm, illustrated in Figure 12. The arm is moved in a trajectory that approaches the object from 0.45 to 1 m, and traverses a solid arc of from approximately 30° to 70° of elevation, and $-60°$ to 60° of azimuth (see video `data_capturing.mp4`). The head stays approximately pointed towards the center of the planar target containing the AprilTags.

The head consists of a Stereolabs ZED stereo camera and

Figure 11: Illustration of our method of using marker to replace the transparent objects with its opaque twin. The white plastic marker was produced by 3D printing.

Figure 12: Robot configuration. Left: we mount both ZED stereo camera (top) and Microsoft Azure Kinect camera (bottom) on the end-effector of the robot, with a plastic fixture produced by 3D printing. Right: Franka Panda arm used to capture data.

a Microsoft Kinect Azure RGBD device.[1] We use the ZED to capture dual synchronized RGB images at 1280×720 resolution, with a baseline of 0.12 m. The camera parameters are calibrated at the factory, and correct for distortion and stereo geometry, with the rectified stereo pair having horizontal epipolar lines. The FOV for the camera is 90(H)×60(V) degrees, fairly wide angle, which introduces perspective distortion at the edges of the images; many other datasets use narrower FOV to avoid this distortion, but we think it is important for the model to deal with it.

The Kinect Azure device is mounted just above the ZED, so the lenses are as close as possible. It consists of an RGB camera, and a time-of-flight depth camera offset by about 2cm. We operate the depth camera in wide-angle mode, 120(H)×120(V) degree FOV, with a resolution of 1024×1024. For the RGB camera, which is synchronized with the depth camera, we capture images at 1280×960, and the FOV is 90(H)×59(V). As with the ZED, distortion and geometry parameters are calibrated at the factory.

Note that the depth camera has several sources of error, including up to 11 mm of systematic error, and a random error standard deviation of 17 mm. Additionally, multi-path interference, especially in corners, can lead to larger distortions; and low-angle incidence often causes dropouts. Figures 1 (in the main paper) and 18 show examples of the depth sensor output. The two devices are not time-synchronized; alignment of depth to the stereo images is done with the method described in Subsection C.3.

Each object is placed in various positions on a background on the planar target board (Figure 19) and the robot is activated, capturing a video of some 400 images in stereo and 200 images in RGBD (slower frame rate). Then, the opaque twin is substituted as shown in Figure 11, and another scan is completed to capture opaque depth. We save all stereo and RGBD images from these scans, to be processed as described below.

C.2. Camera Pose Estimation

To correspond the different camera views, we determine poses from the images of AprilTags on the target board. First, the image coordinates of the corners of the ArpilTags are extracted using publicly-available software from the University of Michigan[2]. Given the known 3D positions of the tags on the board, the PnP algorithm of OpenCV is used to compute the camera pose relative to the frame of the target board.

The board contains eight AprilTags along the borders (Figure 19). We reject any images in which fewer than 3 tags are correctly detected. The mean number of detected tags on a trajectory is 6.1 for the left stereo camera, and 5.9 for the RGB camera of the Kinect (it has a smaller FOV), assuring a robust estimation of the pose. In reprojecting the target AprilTag points back to the camera at its estimated pose, we can compute the RMSE in pixels for the pose estimation over a trajectory. Doing this for all 600 trajectories yields the statistics in Table 7, with the average RMSE at 1.21 pixels for the left stereo camera and 1.30 pixels for the Kinect. We use these values in analyzing the errors in depth warping and 3D keypoint estimation.

C.3. Depth Image Warping

Since the depth images are acquired from a different viewpoint than the stereo images, they must be warped to

[1] Hardware specifications for the ZED are at https://www.stereolabs.com/zed/, and for the Kinect Azure are in https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification.

[2] https://april.eecs.umich.edu/apriltag/.

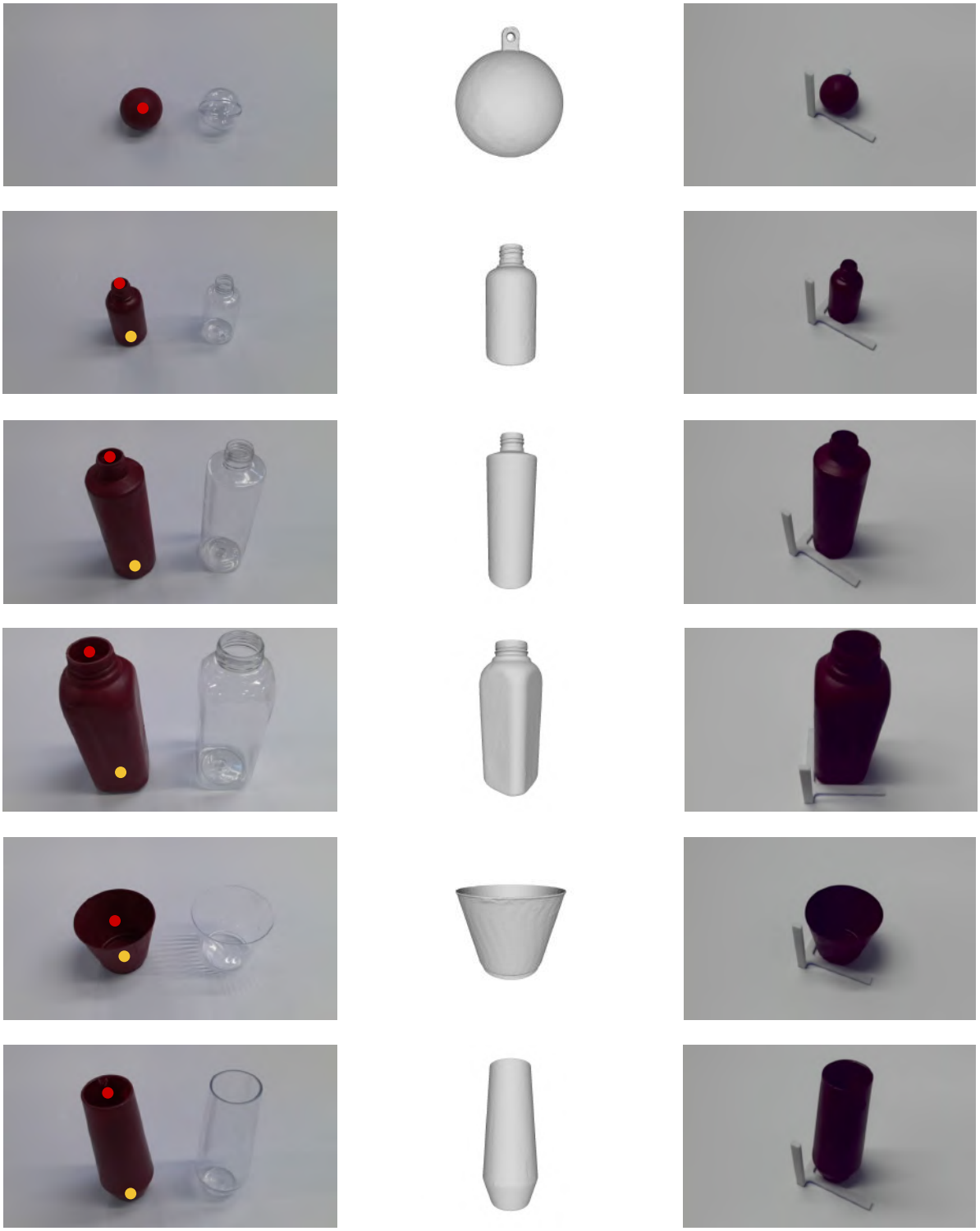

Figure 13: Visualization of ball_0, bottle_0, bottle_1, bottle_2, cup_0, cup_1 from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red and yellow to mark keypoint 1 and 2.

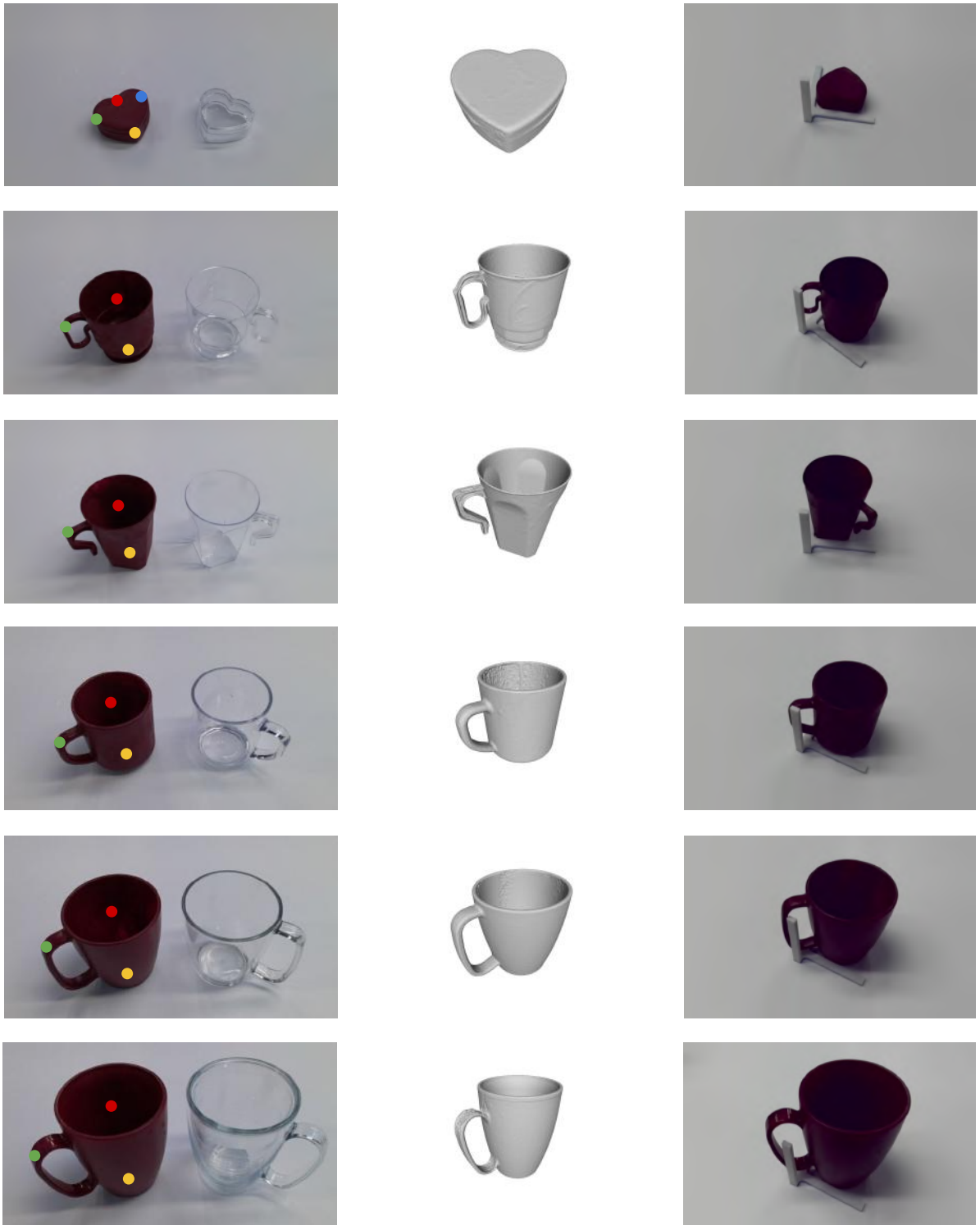

Figure 14: Visualization of heart$_0$, mug$_0$, mug$_1$, mug$_2$, mug$_3$, mug$_4$ from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red, yellow, green and blue to mark keypoint 1, 2, 3 and 4. For heart$_0$, keypoints 3 and 4 are symmetric.

Figure 15: Visualization of mug_5, mug_6, mug_7, mug_8, sakura_0, shovel_0 from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red, yellow, green, blue and purple to mark keypoint 1, 2, 3, 4 and 5. For sakura_0, all its five keypoints are symmetric.

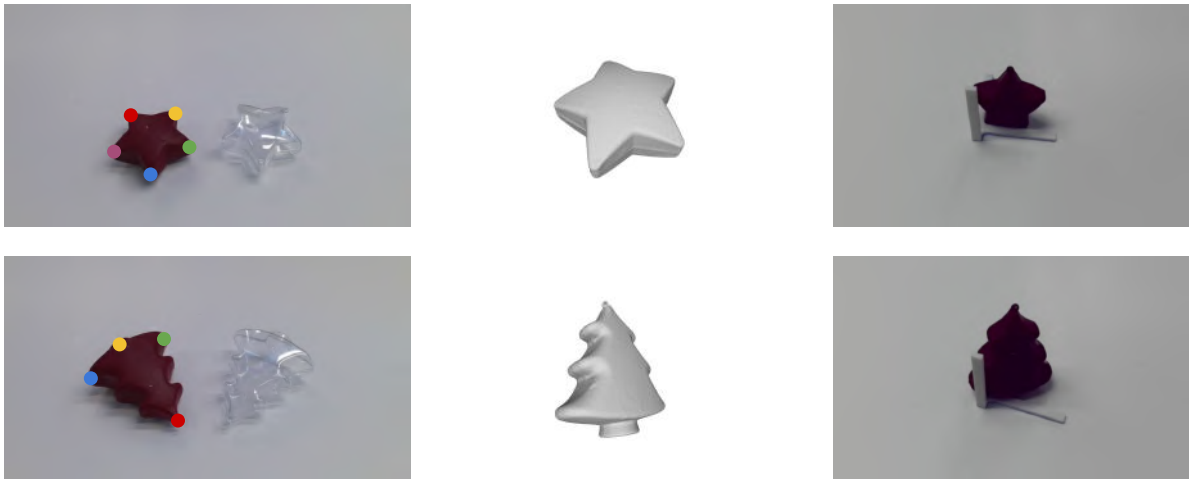

Figure 16: Visualization of star_0 and tree_0 from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red, yellow, green, blue and purple to mark keypoint 1, 2, 3, 4 and 5. For star_0, all its five keypoints are symmetric. For tree_0, keypoints 3 and 4 are symmetric.

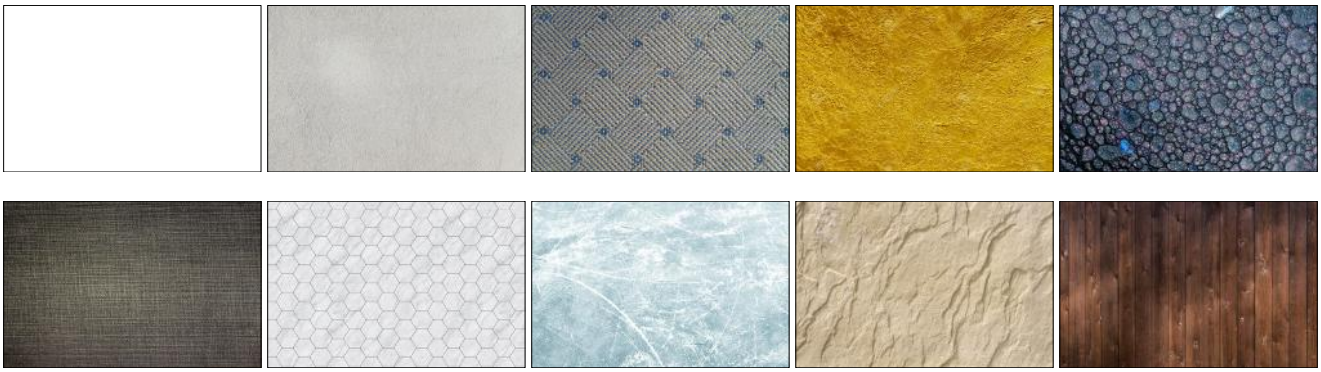

Figure 17: Background textures used in the dataset.

reprojection error (px)	RMSE mean	RMSE std
left stereo	1.21	0.51
Kinect RGB	1.30	0.387

Table 7: Reprojection errors for camera pose estimates (pixels). RMSE is computed over a trajectory; average and standard deviation over a set of 600 trajectories.

register with the latter (we use the left stereo image as the reference image). The depth image, along with the depth camera parameters, gives a 3D point for every pixel, which can then be transformed to a different camera frame and reprojected to form a new depth image registered with that camera. There are several steps to warping the depth image:

1. Remove distortion. Convert 1024×1024 depth image to a 1024×1024 depth image of an ideal pinhole camera. This is a standard image warping operation; we use OpenCV's `undistort` function with the factory-provided calibration parameters.

2. Find the nearest viewpoint. Since the devices are not synchronized, and the images could have been captured on different scans (opaque vs. transparent object), we find the left stereo viewpoint that is closest to the depth image viewpoint. Since the cameras are all registered to a common world view, the target board, it is possible to do this. The viewpoints should be close in both position and orientation. To achieve this, we look at two 3D points, 1 meter ahead of and behind the depth camera. The left stereo camera whose similar points are closest in the world frame is chosen.

3. Compute transform chain. There are three relevant poses: depth camera (${}^{depth}T_{world}$), depth camera

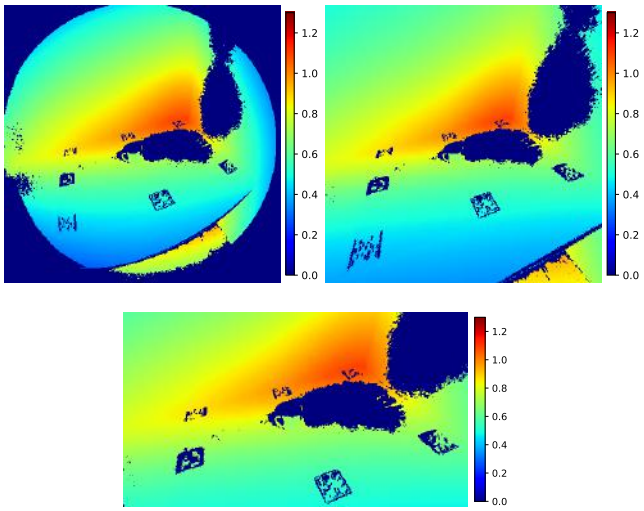

Figure 18: Left: raw depth image. Right: rectified depth image. Bottom: depth image warped to left stereo image.

to Kinect RGB camera, from its known calibration (${}^{rgb}T_{depth}$), and left stereo camera (${}^{left}T_{world}$). The transform from the depth camera to the left stereo camera is the chain: ${}^{left}T_{world}\ {}^{world}T_{rgb}\ {}^{rgb}T_{depth}$.

4. Warp depth to left image. Each point in the rectified depth image is converted to a 3D point in the camera frame, then transformed to the left stereo camera frame via the above transform. Then it is projected onto an image with the same camera parameters as the left image. Z-buffering assures that closer points overlay further ones. We also interpolate pixels in the original depth image to eliminate quantization holes in the transformed image.

The results are shown in Figure 18.

We can get an idea of the pixel errors in warping the depth data from the camera pose estimation errors (Table 7). The combined RMSE is 1.78 pixels ($= \sqrt{1.21^2 + 1.30^2}$). There are additional errors caused by the transform ${}^{rgb}T_{depth}$, which we assume to be sub-pixel from factory calibration, and hence small relative to pose estimation error. The error in depth of the Kinect camera will also result in a projection error, but again, if the left stereo camera and the depth camera viewpoints are close, these should be small and we ignore them.

C.4. 3D Keypoint Labelling and Error Analysis

Given the pose of multiple cameras looking at the same object on the target in a scan, we can compute 3D keypoints by labeling them on a subset of the 2D images. Using a Farthest Point Sampling (FPS) algorithm [7, 6], we pick 6 images that are farthest from each other in position on the scan. For each image, we label the keypoints in the image. These are the projections of the 3D keypoints on the images. Since we know the camera parameters, a simple least-squares nonlinear estimation finds the 3D keypoints that minimize the squared reprojection errors. Once the keypoints are estimated, they are projected back to the labeled views to find the RMSE in pixels. Any scan that has a keypoint error of more than 5 pixels is rejected. We collected RMSE statistics for the mug_2 object: over all scans, the mean RMSE was 2.28 pixels, and the standard deviation was 0.83 pixels. Gathering a single scan and labeling it takes about 10 minutes of user work, so it is possible to acquire large sets of real-world data.

method	3D error (mm)
Microsoft Azure Kinect depth [3]	17
Our keypoint labelling	**3.4**

Table 8: Comparison of keypoint labelling error.

How accurate are the keypoints that are computed from 2D annotations? Unfortunately it is difficult to answer this analytically, because they are computed from two nonlinear optimizations: camera view pose estimation and 3D point estimation from multiple views. Instead, we use Monte Carlo simulation to run thousands of scenarios that conform to the reprojection error statistics that we gathered for camera and keypoint pose estimates. In each simulation, we randomly chose 4 to 6 views taken from the Farthest Point Sampling (FPS) [7, 6] of poses, and calculated the AprilTag corner projections. We then dithered these projections according to the statistics in Table 7, and re-calculated the poses to get estimated poses. Then, we randomly placed a 3D keypoint in the workspace, and projected it onto the estimated poses. Finally, we dithered these projections according to the keypoint RMSE statistics, and estimated the keypoint. The metric distance between the estimated and ground-truth keypoints gives an error measure. We did this for 10,000 simulations, and calculated the RMSE as 3.4 mm. We compare this to the depth error of Microsoft Azure Kinect in Table 8, and conclude that our method is **at least five times more accurate** than the estimation from depth sensors.

D. Architecture and Training Details

D.1. Keypose Architecture

As noted in the paper, the KeyPose architecture is derived from KeyPointNet [30]. Table 9 lists the layers and their parameters.

Stereo RGB images at a resolution of 180×120 are

[3] Depth error data for the Microsoft Azure Kinect can be found in https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification.

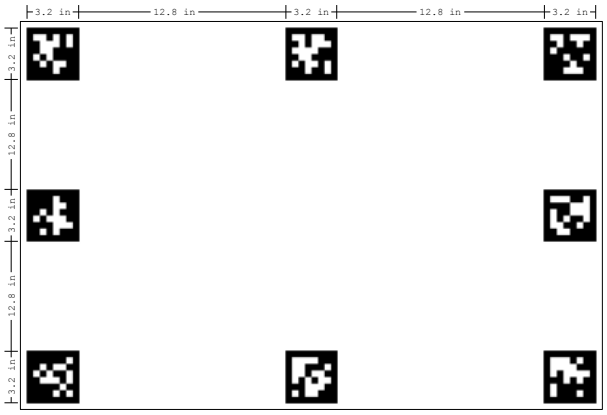

Figure 19: AprilTag board used for capturing data. The positions of AprilTags are fixed therefore global pose can be calculated.

stacked and fed into a set of exponentially-dilated 3×3 convolutions [37] that expands the context for predicting keypoints, while keeping the resolution constant. The exponential series expands the context for each pixel to an area of 64×64 [37]. Repeating this sequence twice ensures an even wider context. After each dilated convolution, we apply batch normalization followed by leaky RELU activation (alpha of 0.1). We also insert $L2$ regularization with a factor of 0.001.

UVD coordinates are extracted either by direct regression to numeric values, or with an integral image. For direct regression, we add two 1x1 convolutions with 64 features and $L2$ regularization, again followed by batch normalization and leaky RELU activation. Then, we have one 1x1 convolution with $3N$ features, where N is the number of keypoints. The output of this convolution is taken directly as the *UVD* values for the keypoints.

For the integral image technique, we add a 3x3 convolution with N features and $L2$ regularization. The output is processed by a spatial softmax to convert it to a probability, and then integrated to find the centroid (and hence the *UV* coordinates), as in IntegralNet [29]. A disparity heatmap is computed by a 3x3 convolution with N features and $L2$ regularization, convolved with the probability map, and the centroid predicts the disparity.

We experimented with various other architectures, including UNet [26] and adding an explicit correlation operator. These did not do better than the dilated CNN.

D.2. Training

As described in the paper, we trained with a batch size of 32 and about 300 epochs for instance training, and 200 epochs for category training, which has more training samples. We used the ADAM optimizer in TensorFlow, with a learning rate of 1×10^{-3}, and successively reducting to 5×10^{-6} by the end. We did not do a systematic hyperparameter search, which might be able to improve results. Based on experience with the training, we use a curriculum that introduces the projection loss when 1/3 of the training is done, and ramps it up fully by 2/3 of the training. The coefficient for projection loss was set at 2.5; more did not improve the results, while less tended to cause higher error in the disparity.

layer #	kernel size	dilations	stride	# of channels
1–7	3	1,1,2,4,8,16,32	1	48 / 64
8–15	3	1,1,2,4,8,16,32	1	48 / 64
prob	3	1	1	N
disp	3	1	1	N
regress	1	1,1,1	1	64, 64, $3N$

Table 9: Architecture of Keypose Early Fusion. The number of channels is 48 for instance models, and 64 for category models. N is the number of keypoints.

The network has a tendency to overfit unless care is taken during training in augmenting the data. We performed both geometric and photometric augmentations. For geometry, rotating the view around the camera center yields a new realistic view of the object, and can be performed via 2D warping operations on the image, without knowing the 3D geometry of the scene. However, we are constrained by stereo geometry, as we want to keep the epipolar constraints along horizontal lines, allowing the network to determine correlation between the left and right images. Rotation around the camera X-axis, scaling and shear along the image Y-axis, and flipping the image Y axis are operations that preserve this constraint. We implemented X-axis rotation, using a random value in the interval $[-5°, +5°]$. We have not yet tried scaling and shear operations.

Another transformation that doubles the size of the dataset is *mirroring*. In this operation, the right and left images are swapped, while preserving epipolar geometry. Note that, if we rotate the stereo camera 180 degrees around the center between the two cameras, turning it upside-down, the new left camera will now see an upside-down version of the right camera image, and the new right camera an upside-down image of the left camera image. Since we prefer to deal with upright images, we can flip the two new images vertically while preserving epipolar geometry. Figure 20 shows a typical example.

For photometric augmentations, we used tensorflow operations to randomize hue, saturation, contrast, and brightness. For hue, we chose a `max_delta` of 0.1. For saturation, the bounds were between 0.6 and 1.2. For contrast, the bounds were between 0.7 and 1.2. For brightness, we chose a `max_delta` of 32/255. We also drop out random elliptical portions of the input images and replace them with background images. Finally, images were normalized by

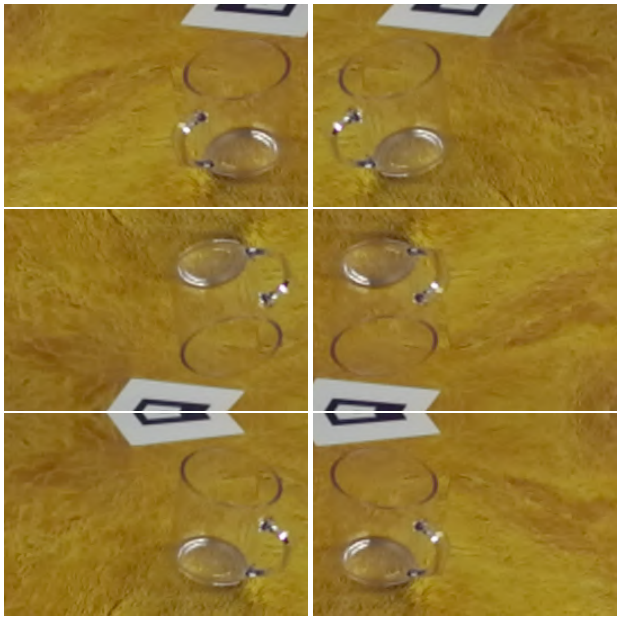

Figure 20: Mirroring stereo data. Top: original left/right images. Middle: Image pair when stereo camera is rotated 180 degrees. Left and right are turned upside down and switched. Bottom: Each image is flipped vertically to give an upright stereo pair, that looks like a mirrored version of the original, preserving epipolar geometry.

subtracting a mean value and scaling by a standard deviation. These values were taken from ImageNet training: the mean values for RGB were $[0.485, 0.456, 0.406]$, and the standard deviation values were $[0.229, 0.224, 0.225]$.

A good measure to track during training and testing is Mean Absolute Error (MAE): the error in metric distance between the predicted and labeled 3D keypoints, in meters. We use the average error rather than mean square error to alleviate undue influence from outliers. Even with all augmentations, the network will overfit, in that training MAE typically goes to around 5mm, while the testing MAE can be several times that, depending on the type of test data (instance vs. category, held-out texture vs. held-out object). Future work will be to find models and training that generalize better.

D.3. Pose From Keypoints

3D keypoints are a flexible way to describe the pose of an object, although they are not always a minimal parameterization. For example, a rigid object without symmetry has 6 DOF, while a minimum of 3 non-collinear keypoints (9 parameters) are needed. But keypoints have the advantage of being able to describe articulated and deformable objects, such as the human hand, although we do not take advantage of that capability in this paper.

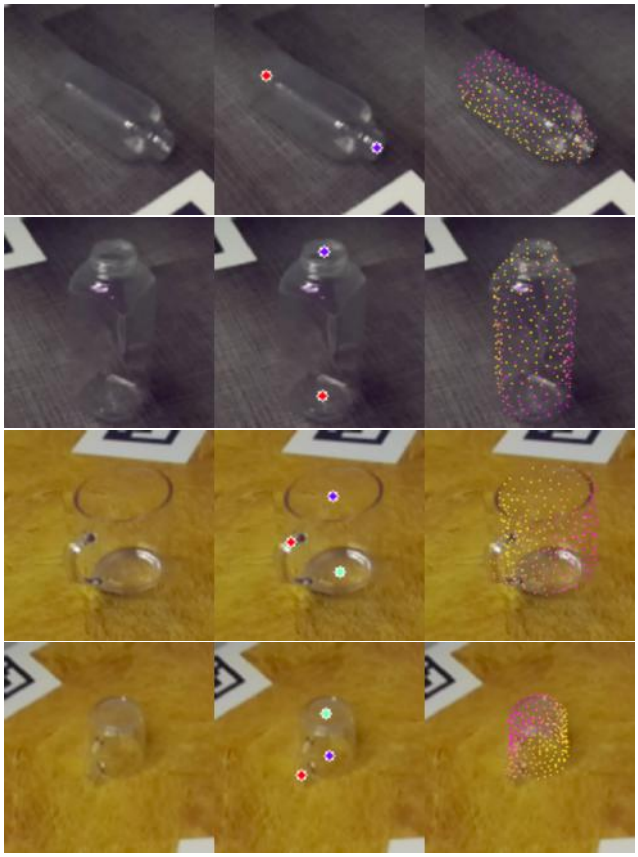

Figure 21: Left: Input left image. Middle: Predicted 3D keypoints. Right: Projected points from the CAD model, for bottle_2 and mug_2.

Since we have CAD models of the transparent objects, we can use the predicted 3D keypoints to align the models to the camera view, and project them into the image. In the CAD models, we have labeled the 3D keypoints so they correspond to the ones labeled in the dataset. Aligning two sets of 3D keypoints, with known correspondence, can be done using the orthogonal Procrustes algorithm [8]. Figure 21 shows the result, for a bottle and mug.

D.4. Model Run Time

The dilated CNN architecture performs inference efficiently, even though it stays at the same resolution as the input, since the number of features does not expand. Typical runtimes for inference on a single sample, using a NVidia Titan V GPU and an i7 desktop, is 3 ms. This does not include the time it would take to find a bounding box for a full detection and pose estimation pipeline.

D.5. Baseline Method Details

We use a variation of DenseFusion [32] as the baseline to compare with our KeyPose. There are two differences

between the variation model and the original DenseFusion model. First, when extracting point clouds in the first stage, the original DenseFusion model assumes knowing the object segmentation masks in RGB images, while the variation model only assumes knowing the same rough detection bounding boxes as KeyPose in order to fairly compare with KeyPose. Therefore, the extracted point clouds are different. Second, instead of regressing 6DoF poses, the variation model directly predicts the locations of the 3D keypoints. We use similar permutation loss to train the variation DenseFusion model.