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

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

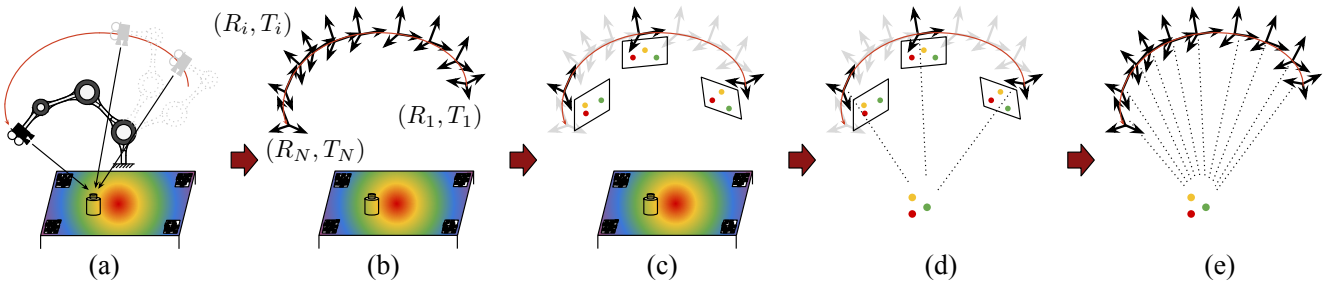

Figure 1: Labeling pipeline. (a) We use robot to scan the object from different views and record stereo-RGB and RGBD sequences; (b) AprilTags groundtruth locations are used to calculate the global pose of each frame (1 through N) in the video based on Perspective-n-Point (PnP) algorithm; (c) Only a few 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.

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 2, 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 4, 5, 6 and 7. 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 4, 5, 6 and 7.

Textures Used in Our Dataset. In our dataset, each object in placed on ten diverse background textures, which are illustrated in Figure 8. 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 1.

Figure 2: 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 3: 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.

C.1. Data Capture Pipeline

Data is captured via a sensor head attached to a Franka Panda arm, illustrated in Figure 3. 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 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, multipath interference, especially in corners, can lead to larger distortions; and low-angle incidence often causes dropouts. Figures 1 (in the main paper) and 9 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 10) 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 2, 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 10). 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

[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/.

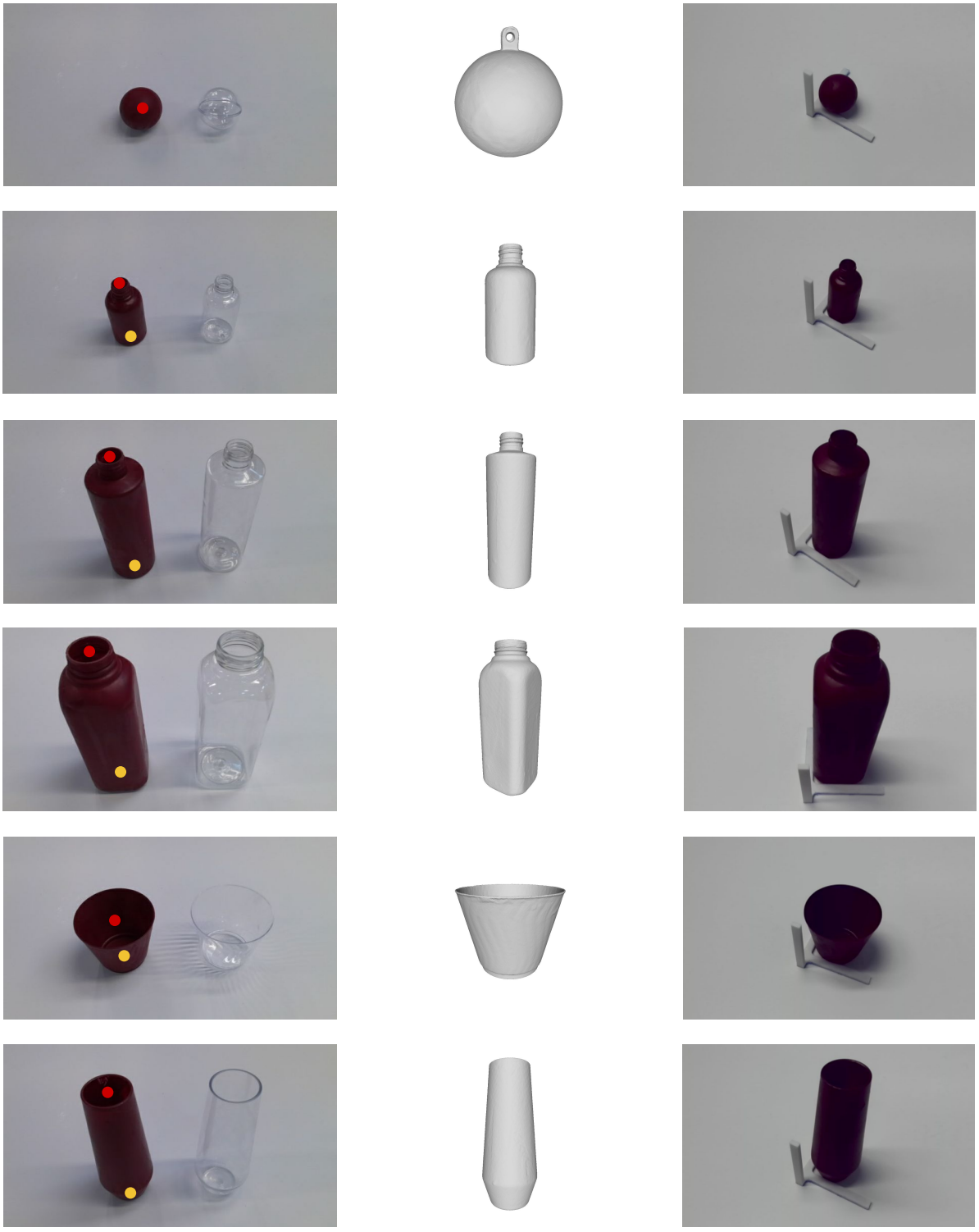

Figure 4: Visualization of ball_0, bottle_0, bottle_1, bottle_2, cup_0, cup_1 from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red and yellow to mark keypoint 1 and 2.

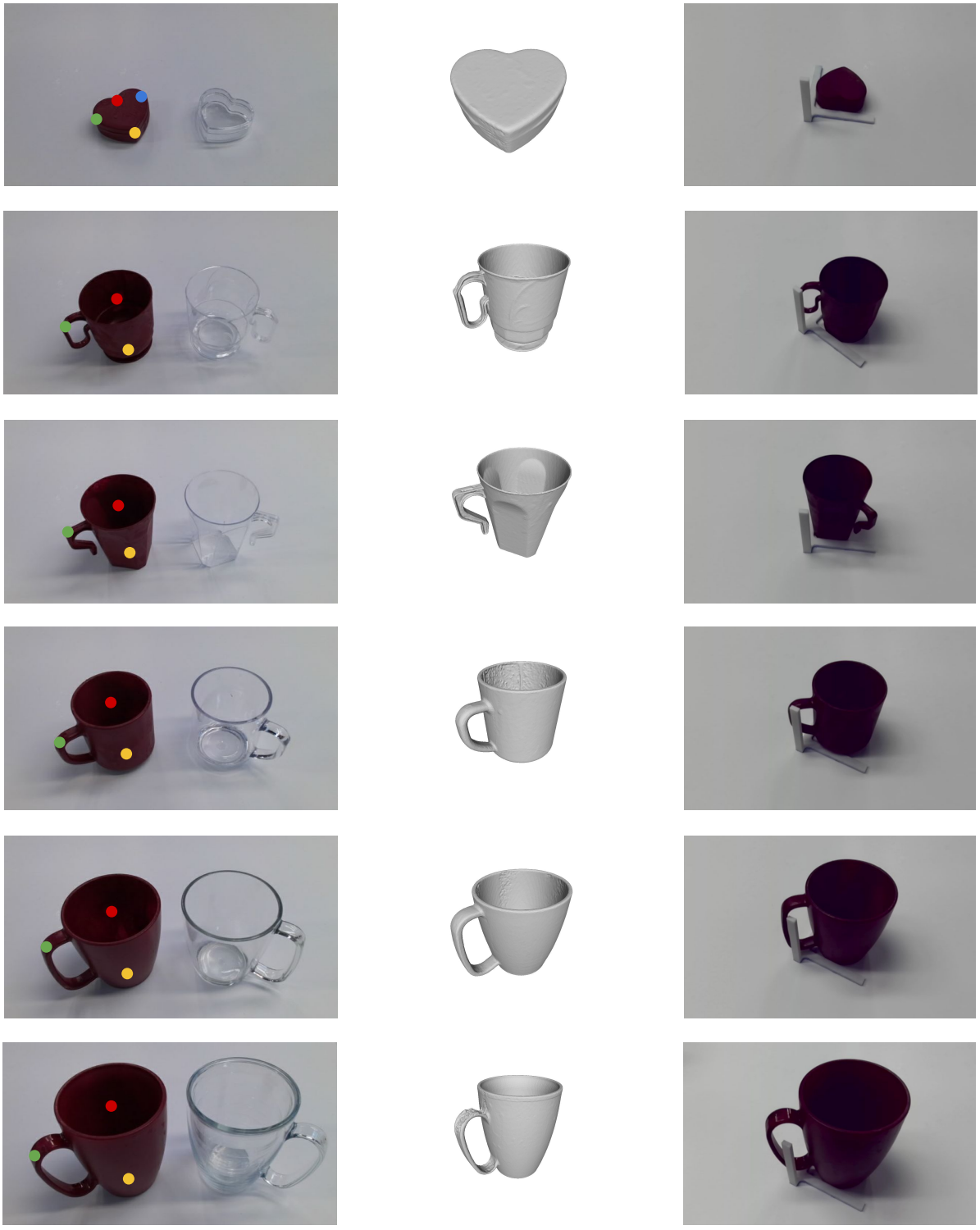

Figure 5: Visualization of heart_0, mug_0, mug_1, mug_2, mug_3, mug_4 from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red, yellow, green and blue to mark keypoint 1, 2, 3 and 4. For heart_0, keypoints 3 and 4 are symmetric.

Figure 6: 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.

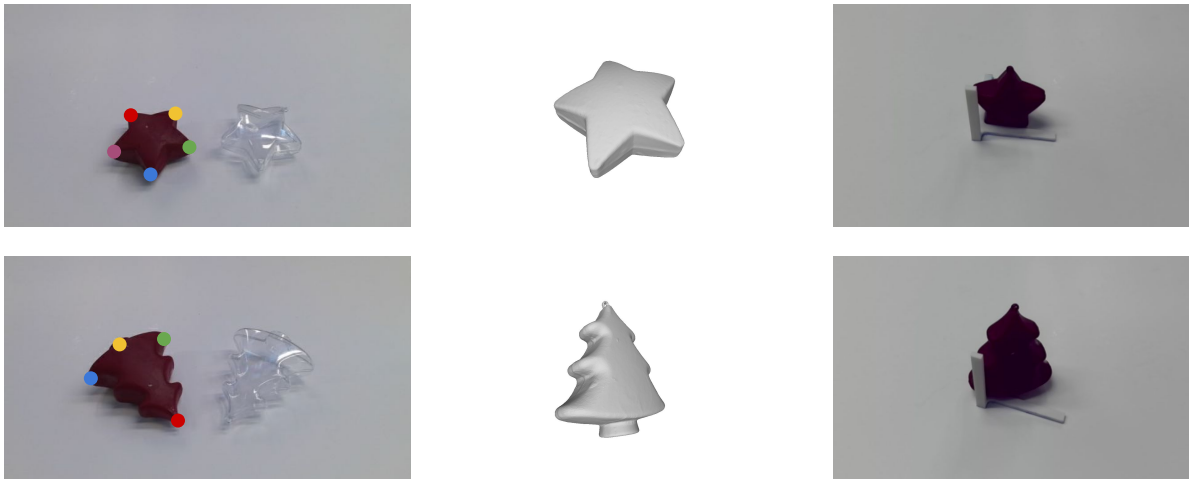

Figure 7: Visualization of star_0 and tree_0 from top to bottom. From left to right in each row: object twin with groundtruth keypoint location, scanned CAD model from opaque object, and aligning the marker to the object. We use red, yellow, green, blue and purple to mark keypoint 1, 2, 3, 4 and 5. For star_0, all its five keypoints are symmetric. For tree_0, keypoints 3 and 4 are symmetric.

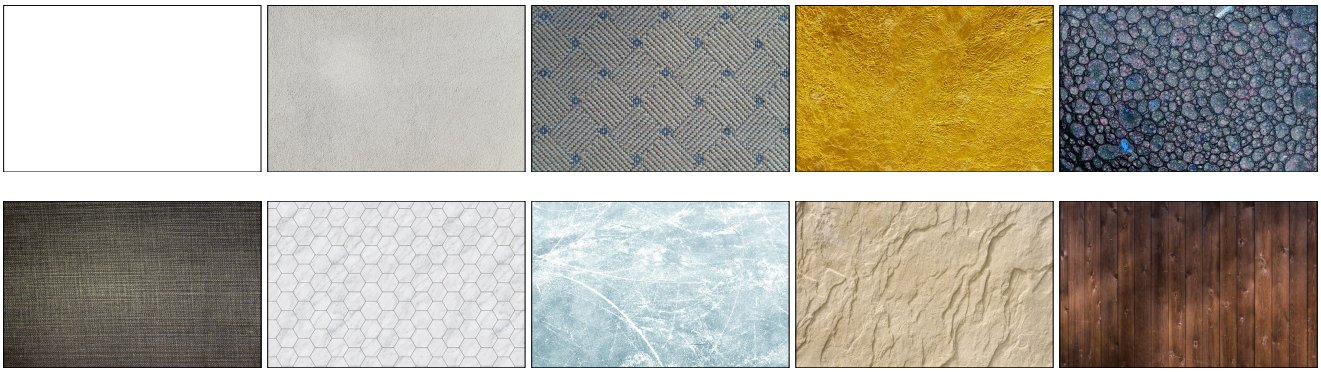

Figure 8: Background textures used in the dataset.

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

Table 1: Reprojection errors for camera pose estimates (pixels). RMSE is computed over a trajectory; average and standard deviation over a set of 600 trajectories.

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 1, 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 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 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 9.

We can get an idea of the pixel errors in warping the depth data from the camera pose estimation errors (Table 1). 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 [2, 1], 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.

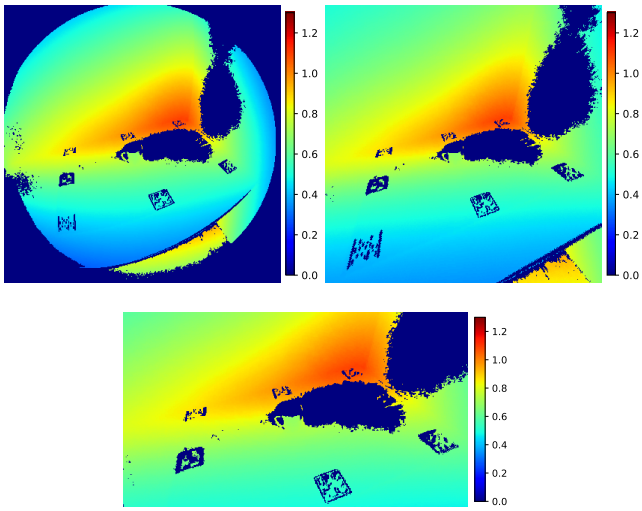

Figure 9: Left: raw depth image. Right: rectified depth image. Bottom: depth image warped to left stereo image.

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

Table 2: 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) [2, 1] of poses, and calculated the AprilTag corner projections. We then dithered these projections according to the statistics in Table 1, 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 2, and conclude that our method is **at least five times more accurate** than the estimation from depth sensors.

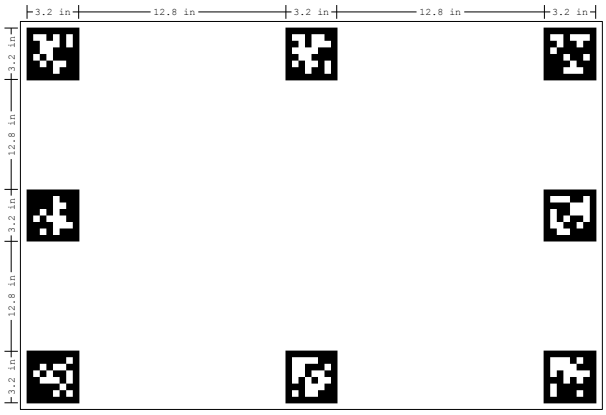

Figure 10: AprilTag board used for capturing data. The positions of AprilTags are fixed therefore global pose can be calculated.

D. Architecture and Training Details

D.1. Keypose Architecture

As noted in the paper, the KeyPose architecture is derived from KeyPointNet [6]. Table 3 lists the layers and their parameters.

Stereo RGB images at a resolution of 180×120 are stacked and fed into a set of exponentially-dilated 3×3 convolutions [8] 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 [8]. 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 [5]. 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.

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

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 3: 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.

We experimented with various other architectures, including UNet [4] 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.

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

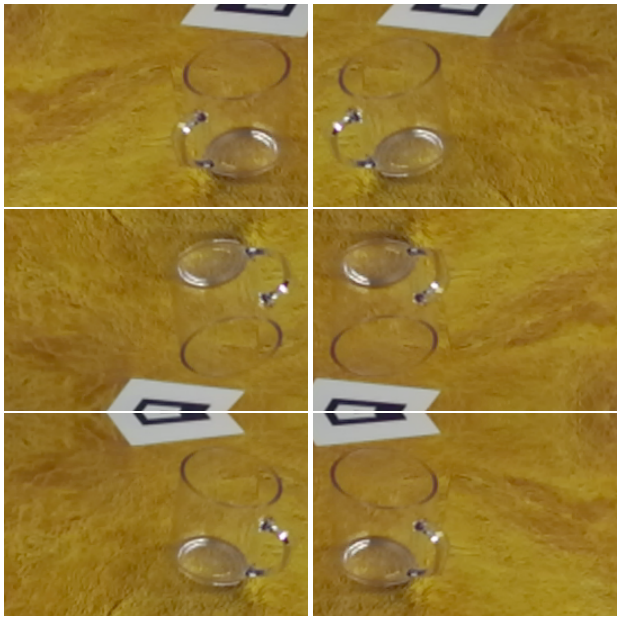

Figure 11: Mirroring stereo data. Top: original left/right images. Middle: Image pair when stereo camera is rotated 180 degrees. Left and right are turned upside down and switched. Bottom: Each image is flipped vertically to give an upright stereo pair, that looks like a mirrored version of the original, preserving epipolar geometry.

vertically while preserving epipolar geometry. Figure 11 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 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.

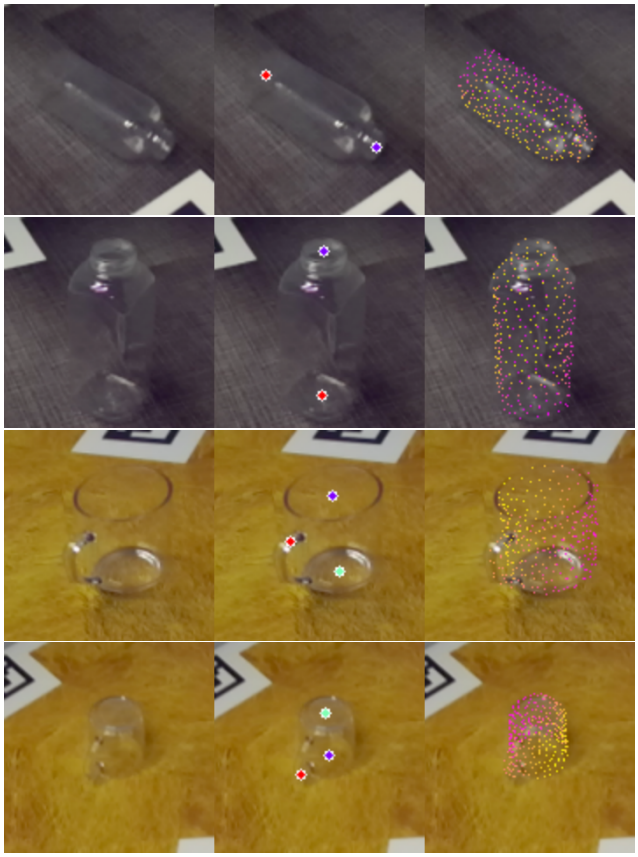

Figure 12: Left: Input left image. Middle: Predicted 3D keypoints. Right: Projected points from the CAD model, for bottle_2 and mug_2.

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.

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 [3]. Figure 12 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 [7] 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.

References

[1] Yuval Eldar, Michael Lindenbaum, Moshe Porat, and Yehoshua Y Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 1997. 7

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

[3] John C Gower. Generalized procrustes analysis. *Psychometrika*, 1975. 9

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015. 8

[5] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral human pose regression. In *ECCV*, 2018. 8

[6] Supasorn Suwajanakorn, Noah Snavely, Jonathan Tompson, and Mohammad Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *NIPS*, 2018. 8

[7] 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. 10

[8] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. 8