Real-Time Seamless Single Shot 6D Object Pose Prediction

Bugra Tekin
EPFL
bugra.tekin@epfl.ch

Sudipta N. Sinha
Microsoft Research
sudipta.sinha@microsoft.com

Pascal Fua
EPFL
pascal.fua@epfl.ch

Abstract

We propose a single-shot approach for simultaneously detecting an object in an RGB image and predicting its 6D pose without requiring multiple stages or having to examine multiple hypotheses. Unlike a recently proposed single-shot technique for this task [11] that only predicts an approximate 6D pose that must then be refined, ours is accurate enough not to require additional post-processing. As a result, it is much faster – 50 fps on a Titan X (Pascal) GPU – and more suitable for real-time processing. The key component of our method is a new CNN architecture inspired by [28, 29] that directly predicts the 2D image locations of the projected vertices of the object's 3D bounding box. The object's 6D pose is then estimated using a PnP algorithm.

For single object and multiple object pose estimation on the LINEMOD *and* OCCLUSION *datasets, our approach substantially outperforms other recent CNN-based approaches [11, 26] when they are all used without post-processing. During post-processing, a pose refinement step can be used to boost the accuracy of these two methods, but at 10 fps or less, they are much slower than our method.*

1. Introduction

Real-time object detection and 6D pose estimation is crucial for augmented reality, virtual reality, and robotics. Currently, methods relying on depth data acquired by RGB-D cameras are quite robust [1, 4, 5, 12, 14]. However, active depth sensors are power hungry, which makes 6D object detection methods for passive RGB images more attractive for mobile and wearable cameras. There are many fast keypoint and edge-based methods [22, 32, 37] that are effective for textured objects. However, they have difficulty handling weakly textured or untextured objects and processing low-resolution video streams, which are quite common when dealing with cameras on wearable devices.

Deep learning techniques have recently been used to address these limitations [11, 26]. BB8 [26] is a 6D object detection pipeline made of one CNN to coarsely segment the object and another to predict the 2D locations of the projections of the object's 3D bounding box given the segmentation, which are then used to compute the 6D pose using a PnP algorithm [17]. The method is effective but slow due to its multi-stage nature. SSD-6D [11] is a different pipeline that relies on the SSD architecture [20] to predict 2D bounding boxes and a very rough estimate of the object's orientation in a single step. This is followed by an approximation to predict the object's depth from the size of its 2D bounding box in the image, to lift the 2D detections to 6D. Both BB8 and SSD-6D require a further pose refinement step for improved accuracy, which increases their running times linearly with the number of objects being detected.

In this paper, we propose a single-shot deep CNN architecture that takes the image as input and directly detects the 2D projections of the 3D bounding box vertices. It is end-to-end trainable and accurate even without any *a posteriori* refinement. And since, we do not need this refinement step, we also do not need a precise and detailed textured 3D object model that is needed by other methods [11, 26]. We only need the 3D bounding box of the object shape for training. This can be derived from other easier to acquire and approximate 3D shape representations.

We demonstrate state-of-the-art accuracy on the LINEMOD dataset [9], which has become a *de facto* standard benchmark for 6D pose estimation. However, we are much faster than the competing techniques by a factor of more than five, when dealing with a single object. Furthermore, we pay virtually no time-penalty when handling several objects and our running time remains constant whereas that of other methods grow proportional to the number of objects, which we demonstrate on the OCCLUSION dataset [1].

Therefore, our contribution is an architecture that yields a fast and accurate one-shot 6D pose prediction without requiring any post-processing. It extends single shot CNN architectures for 2D detection in a seamless and natural way to the 6D detection task. Our implementation is based on YOLO [29] but the approach is amenable to other single-shot detectors such as SSD [20] and its variants.

2. Related Work

We now review existing work on 6D pose estimation ranging from classical feature and template matching methods to newer end-to-end trainable CNN-based methods.

Classical methods. Traditional RGB object instance recognition and pose estimation works used local keypoints and feature matching. Local descriptors needed by such methods were designed for invariance to changes in scale, rotation, illumination and viewpoints [22, 32, 37]. Such methods are often fast and robust to occlusion and scene clutter. However, they only reliably handle textured objects in high resolution images [16]. Other related methods include 3D model-based registration [18, 21, 36], Hausdorff matching [10], oriented Chamfer matching for edges [19] and 3D chamfer matching for aligning 3D curve-based models to images [27].

RGB-D methods. The advent of commodity depth cameras has spawned many RGB-D object pose estimation methods [1, 4, 5, 12, 15, 24, 33, 40]. For example, Hinterstoisser proposed template matching algorithms suitable for both color and depth images [8, 9]. Rios et al. [31] extended their work using discriminative learning and cascaded detections for higher accuracy and efficiency respectively. RGB-D methods were used on indoor robots for 3D object recognition, pose estimation, grasping and manipulation [4, 5, 6, 14, 15, 41]. Brachmann et al. [1] proposed using regression forests to predict dense object coordinates, to segment the object and recover its pose from dense correspondences. They also extended their method to handle uncertainty during inference and deal with RGB images [2]. Zach et al. [39] explored fast dynamic programming based algorithms for RGB-D images.

CNN-based methods. In recent years, research in most pose estimation tasks has been dominated by CNNs. Techniques such as Viewpoints and Keypoints [35] and Render for CNN [34] cast object categorization and 3D pose estimation into classification tasks, specifically by discretizing the pose space. In contrast, PoseNet [13] proposes using a CNN to directly regress from an RGB image to a 6D pose, albeit for camera pose estimation, a slightly different task. Since PoseNet outputs a translational and a rotational component, the two associated loss terms have to be balanced carefully by tuning a hyper-parameter during training.

To avoid this problem, the newer PoseCNN architecture [38] is trained to predict 6D object pose from a single RGB image in multiple stages, by decoupling the translation and rotation predictors. A geodesic loss function more suitable for optimizing over 3D rotations have been suggested in [23]. Another way to address this issue has recently emerged. In [11, 26], the CNNs do not directly predict object pose. Instead, they output 2D coordinates, 2D masks, or discrete orientation predictions from which the 6D pose can be inferred. Because all the predictions are in the 2D image, the problem of weighting different loss terms goes away. Also training becomes numerically more stable, resulting in better performance on the LINEMOD dataset [9]. We also adopt this philosophy in our work.

In parallel to these developments, on the 2D object detection task, there has been a progressive trend towards single shot CNN frameworks as an alternative to two-staged methods such as Faster-RCNN [30] that first find a few candidate locations in the image and then classifies them as objects or background. Recently, single shot architectures such as YOLO [28, 29] and SSD [20] have been shown to be fast and accurate. SSD has been extended to predict the object's identity, its 2D bounding box in the image and a discrete estimate of the object's orientation [11, 25]. In this paper, we go beyond such methods by extending a YOLO-like architecture [29] to directly predict a few 2D coordinates from which the full 6D object pose can be accurately recovered.

3. Approach

With our goal of designing an end-to-end trainable network that predicts the 6D pose in real-time, we were inspired by the impressive performance of single shot 2D object detectors such as YOLO [28, 29]. This led us to design the CNN architecture [28, 29] shown in Fig. 1. We designed our network to predict the 2D projections of the corners of the 3D bounding box around our objects. The main insight was that YOLO was originally designed to regress 2D bounding boxes and to predict the projections of the 3D bounding box corners in the image, a few more 2D points had to be predicted for each object instance in the image. Then given these 2D coordinates and the 3D ground control points for the bounding box corners, the 6D pose can be calculated algebraically with an efficient PnP algorithm [17]. BB8 [26] takes a similar approach. However, they first find a 2D segmentation mask around the object and present a cropped image to a second network that predicts the eight 2D corners in the image. We now describe our network architecture and explain various aspects of our approach in details.

3.1. Model

We formulate the 6D pose estimation problem in terms of predicting the 2D image coordinates of virtual 3D control points associated with the 3D models of our objects of interest. Given the 2D coordinate predictions, we calculate the object's 6D pose using a PnP algorithm. We parameterize the 3D model of each object with 9 control points. For these control points, we select the 8 corners of the tight 3D bounding box fitted to the 3D model, similar to [26]. In addition, we use the centroid of the object's 3D model as the 9th point. This parameterization is general and can be

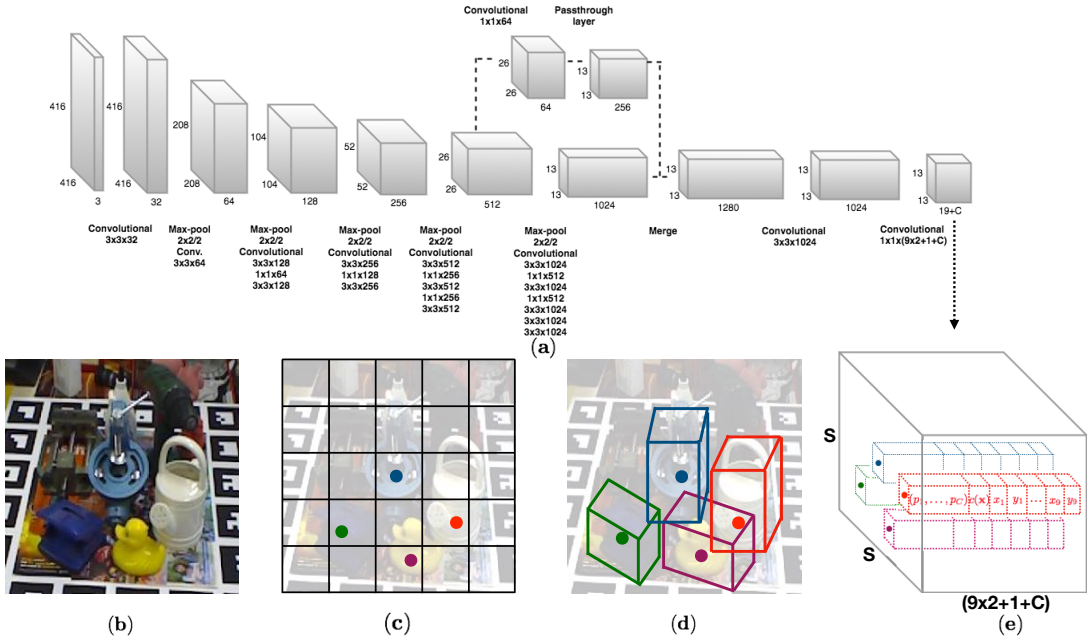


Figure 1. Overview: (a) The proposed CNN architecture. (b) An example input image with four objects. (c) The $S \times S$ grid showing cells responsible for detecting the four objects. (d) Each cell predicts 2D locations of the corners of the projected 3D bounding boxes in the image. (e) The 3D output tensor from our network, which represents for each cell a vector consisting of the 2D corner locations, the class probabilities and a confidence value associated with the prediction.

used for any rigid 3D object with arbitrary shape and topology. In addition, these 9 control points are guaranteed to be well spread out in the 2D image and could be semantically meaningful for many man-made objects.

Our model takes as input a single full color image, processes it with a fully-convolutional architecture shown in Figure 1(a) and divides the image into a 2D regular grid containing $S \times S$ cells as shown in Figure 1(c). In our model, each grid location in the 3D output tensor will be associated with a multidimensional vector, consisting of predicted 2D image locations of the 9 control points, the class probabilities of the object and an overall confidence value. At test time, predictions at cells with low confidence values, ie. where the objects of interest are not present, will be pruned.

The output target values for our network are stored in a 3D tensor of size $S \times S \times D$ visualized in Fig. 1(e). The target values for an object at a specific spatial cell location $i \in S \times S$ is placed in the i-th cell in the 3D tensor in the form of a D dimensional vector $\mathbf{v}_i$. When N objects are present in different cells, we have N such vectors, $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ in the 3D tensor. We train our network to predict these target values. The 9 control points in our case are the 3D object model's center and bounding box corners but could be defined in other ways as well. To train our network, we only need to know the 3D bounding box of the object, not a detailed mesh or an associated texture map.

As in YOLO, it is crucial that a trained network is able to predict not only the precise 2D locations but also high confidence values in regions where the object is present and low confidence where it isn't present. In case of 2D object detection, YOLO uses for its confidence values, an intersection over union (IoU) score associated with the predicted (and true 2D rectangles) in the image. In our case, the objects are in 3D and to compute an equivalent IoU score with two arbitrary cuboids, we would need to calculate a 3D convex hull corresponding to their intersections. This would be tedious and would slow down training, as also analyzed in our supplemental material. Therefore, we take a different approach. We model the predicted confidence value using a confidence function shown in Figure 2. The confidence function, $c(\mathbf{x})$, returns a confidence value for a predicted 2D point denoted by $\mathbf{x}$ based on its distance $D_T(\mathbf{x})$ from the ground truth i.e. target 2D point. Formally, we define the confidence function $c(\mathbf{x})$ as follows:

$$c(\mathbf{x}) = \begin{cases} e^{\alpha(1-\frac{D_T(\mathbf{x})}{d_{th}})}, & \text{if } D_T(\mathbf{x}) < d_{th} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

The distance $D_T(\mathbf{x})$ is defined as the 2D Euclidean distance in the image space. To achieve precise localization with this function, we choose a sharp exponential function with a cut-off value d_{th} instead of a monotonically decreasing linear function. The sharpness of the exponential function is defined by the parameter α. In practice, we apply the confidence function to all the control points and calculate the mean value and assign it as the confidence. As mentioned earlier, we also predict C conditional class prob-

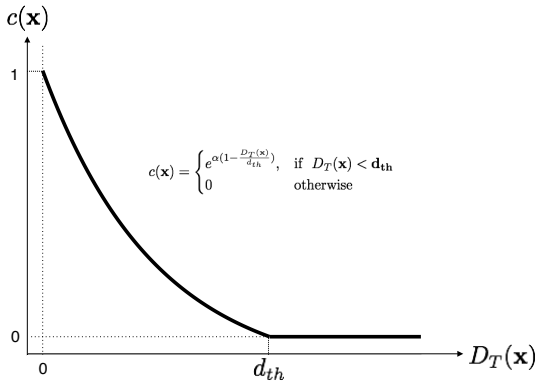


Figure 2. Confidence $c(\mathbf{x})$ as a function of the distance $D_T(\mathbf{x})$ between a predicted point and the true point.

abilities at each cell. The class probability is conditioned on the cell containing an object. Overall, our output 3D tensor depicted in Figure 1(**e**) has dimension $S \times S \times D$, where the 2D spatial grid corresponding to the image dimensions has $S \times S$ cells and each such cell has a D dimensional vector. Here, $D = 9 \times 2 + C + 1$, because we have 9 (x_i, y_i) control points, C class probabilities and one confidence value.

Our network architecture follows the fully convolutional YOLO v2 architecture [29]. Thus, our network has 23 convolutional layers and 5 max-pooling layers. Similar to YOLO v2, we choose $S = 13$ and have a 13×13 2D spatial grid on which we make our predictions. We also allow higher layers of our network to use fine-grained features by adding a passthrough layer. Specifically, we bring features from an earlier layer at resolution 26×26, apply batch normalization and resize the input image during training on-the-fly. As the network downsamples the image by a factor of 32, we change the input resolution to a multiple of 32 randomly chosen from the set $\{320, 352, \ldots, 608\}$ to be robust to objects of different size.

3.2. Training Procedure

Our final layer outputs class probabilities, (x, y) coordinate locations for the control points, and the overall confidence score. During training, this confidence value is computed on the fly using the function defined in Eq. 1 to measure the distance between the current coordinate predictions and the ground-truth, $D_T(\mathbf{x})$. We predict offsets for the 2D coordinates with respect to (c_x, c_y), the top-left corner of the associated grid cell. For the centroid, we constrain this offset to lie between 0 and 1. However, for the corner points, we do not constrain the network's output as those points should be allowed to fall outside the cell. The predicted control point (g_x, g_y) is defined as

$$g_x = f(x) + c_x \tag{2}$$

$$g_y = f(y) + c_y \tag{3}$$

where $f(\cdot)$ is chosen to be a 1D sigmoid function in case of the centroid and the identity function in case of the eight corner points. This has the effect of forcing the network to first find the approximate cell location for the object and later refine its eight corner locations. We minimize the following loss function to train our complete network.

$$\mathcal{L} = \lambda_{pt}\mathcal{L}_{pt} + \lambda_{conf}\mathcal{L}_{conf} + \lambda_{id}\mathcal{L}_{id} \tag{4}$$

Here, the terms $\mathcal{L}_{pt}$, $\mathcal{L}_{conf}$ and $\mathcal{L}_{id}$ denote the coordinate, confidence and the classification loss, respectively. We use mean-squared error for the coordinate and confidence losses, and cross entropy for the classification loss. As suggested in [28], we downweight the confidence loss for cells that don't contain objects by setting λ_{conf} to 0.1. This improves model stability. For cells that contain objects, we set λ_{conf} to 5.0. We set λ_{pt} and λ_{id} simply to 1.

When multiple objects are located close to each other in the 3D scene, they are more likely to appear close together in the images or be occluded by each other. In these cases, certain cells might contain multiple objects. To be able to predict the pose of such multiple objects that lie in the same cell, we allow up to 5 candidates per cell and therefore predict five sets of control points per cell. This essentially means that we assumed that at most 5 objects could occlude each other in a single grid cell. This is a reasonable assumption to make in practical pose estimation scenarios. As in [29], we precompute with k-means, five anchor boxes that define the size, ie. the width and height of a 2D rectangle tightly fitted to a masked region around the object in the image. During training, we assign whichever anchor box has the most similar size to the current object as the responsible one to predict the 2D coordinates for that object.

3.3. Pose Prediction

We detect and estimate the pose of objects in 6D by invoking our network only once. At test time, we estimate the class-specific confidence scores for each object by multiplying the class probabilities and the score returned by the confidence function. Each grid cell produces predictions in one network evaluation and cells with predictions with low confidence are pruned using a confidence threshold. For large objects and objects whose projections lie at the intersection of two cells, multiple cells are likely to predict highly confident detections. To obtain a more robust and well localized pose estimate, we inspect the cells in the 3×3 neighborhood of the cell which has the maximum confidence score. We combine the individual corner predictions of these adjacent cells by computing a weighted average of the individual detections, where the weights used are the confidence scores of the associated cells.

At run-time, the network gives the 2D projections of the object's centroid and corners of its 3D bounding box along with the object identity. We estimate the 6D pose from the correspondences between the 2D and 3D points using a Perspective-n-Point (PnP) pose estimation method [17].

In our case, PnP uses only 9 such control point correspondences and provides an estimate of the 3D rotation $\mathbf{R}$ and 3D translation $\mathbf{t}$ of the object in camera coordinates.

4. Implementation Details

We initialize the parameters of our network by training the original network on the ImageNet classification task. As the pose estimates in the early stages of training are inaccurate, the confidence values computed using Eq. 1 are initially unreliable. To remedy this, we pretrain our network parameters by setting the regularization parameter for confidence to 0. Subsequently, we train our network by setting λ_{conf} to 5 for the cells that contain an object, and to 0.1 otherwise, to have more reliable confidence estimates in the early stages of the network. In practice, we set the sharpness of the confidence function α to 2 and the distance threshold to 30 pixels. We use stochastic gradient descent for optimization. We start with a learning rate of 0.001 and divide the learning rate by 10 at every 100 epochs. To avoid overfitting, we use extensive data augmentation by randomly changing the hue, saturation and exposure of the image by up to a factor of 1.5. We also randomly scale and translate the image by up to a factor of 20% of the image size. Our implementation is based on PyTorch. We will make our code publicly available for the sake of reproducibility.

5. Experiments

We first evaluate our method for estimating the 6D pose of single objects and then we evaluate it in the case where multiple objects are present in the image. We use the same datasets and evaluation protocols as in [2, 11, 26], which we review below. We then present and compare our results to the state of the art methods.

5.1. Datasets

We test our approach on two datasets that were designed explicitly to benchmark 6D object pose estimation algorithms. We describe them briefly below.

LineMod [9] has become a *de facto* standard benchmark for 6D object pose estimation of textureless objects in cluttered scenes. The central object in each RGB image is assigned a ground-truth rotation, translation, and ID. A full 3D mesh representing the object is also provided. There are 15783 images in LINEMOD for 13 objects. Each object features in about 1200 instances.

OCCLUSION [1] is a multi-object detection and pose estimation dataset that contains additional annotations for *all* objects in a subset of the LINEMOD images. As its name suggests, several objects in the images are severely occluded due to scene clutter, which makes pose estimation extremely challenging. With the exception of [11, 26], it has primarily been used to test algorithms that require depth images.

Method	w/o Refinement			w/ Refinement	
Object	Brachmann [2]	BB8 [26]	OURS	Brachmann [2]	BB8 [26]
Ape	-	***95.3***	92.10	85.2	**96.6**
Benchvise	-	80.0	**95.06**	67.9	90.1
Cam	-	80.9	**93.24**	58.7	86.0
Can	-	84.1	**97.44**	70.8	91.2
Cat	-	97.0	**97.41**	84.2	98.8
Driller	-	74.1	***79.41***	73.9	**80.9**
Duck	-	81.2	**94.65**	73.1	92.2
Eggbox	-	87.9	**90.33**	83.1	91.0
Glue	-	89.0	**96.53**	74.2	92.3
Holepuncher	-	90.5	**92.86**	78.9	95.3
Iron	-	78.9	***82.94***	83.6	**84.8**
Lamp	-	74.4	**76.87**	64.0	75.8
Phone	-	77.6	**86.07**	60.6	85.3
Average	69.5	83.9	**90.37**	73.7	89.3

Table 1. Comparison of our approach with state-of-the-art algorithms on LINEMOD in terms of 2D reprojection error. We report percentages of correctly estimated poses. In Tables 1, 2 and 4 **bold face** numbers denote the best overall methods, ***bold italic*** numbers denote the best methods among those that do not use refinement as opposed to the ones that use, if different. Note that even though we do not rely on the knowledge of a detailed 3D object model our method consistently outperforms the baselines.

5.2. Evaluation Metrics

We use three standard metrics to evaluate 6D pose accuracy, namely – 2D reprojection error, IoU score and average 3D distance of model vertices (referred to as ADD metric) as in [2, 11, 26]. In all cases, we calculate the accuracy as the percentage of *correct* pose estimates for certain error thresholds.

When using the reprojection error, we consider a pose estimate to be correct when the mean distance between the 2D projections of the object's 3D mesh vertices using the estimate and the ground truth pose is less than 5 pixels [2]. This measures the closeness of the true image projection of the object to that obtained by using the estimated pose. This metric is suitable for augmented reality applications.

To compute the IoU score, we measure the overlap between the projections of the 3D model given the ground-truth and predicted pose and accept a pose as correct if the overlap is larger than 0.5, as in [11].

When comparing 6D poses using the ADD metric, we take a pose estimate to be correct if the mean distance between the true coordinates of 3D mesh vertices and those estimated given the pose is less than 10% of the object's diameter [9]. For most objects, this is approximately a 2cm threshold but for smaller objects, such as *ape*, the threshold drops to about 1cm. For rotationally symmetric objects whose pose can only be computed up to one degree of rotational freedom, we modify slightly the metric as in [2, 9]

and compute

$$s = \frac{1}{|\mathcal{M}|} \sum_{x_1 \in \mathcal{M}} \min_{\mathcal{M}} \|(\mathbf{R}\mathbf{x} + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})\| \,, \quad (5)$$

where $(\mathbf{R}, \mathbf{t})$ are the ground-truth rotation and translation, $(\hat{\mathbf{R}}, \hat{\mathbf{t}})$ the predicted ones, and $\mathcal{M}$ the vertex set of the 3D model. We use this metric when evaluating the pose accuracy for the rotationally invariant objects, *eggbox* and *glue* as in [2, 9].

5.3. Single Object Pose Estimation

We first estimate the 6D pose of the central object in the RGB only LINEMOD images, without reference to the depth ones. We compare our approach to those of [2, 11, 26], which operate under similar conditions.

In this dataset, the training images are selected such that the relative orientation between corresponding pose annotations are larger than a threshold. As in [2, 11, 26], to avoid being influenced by the scene context and overfitting to the background, we segment the training images using the segmentation masks provided with the dataset and replace the background by a random image from the PASCAL VOC dataset [7].

We use exactly the same training/test splits as in [26]. We report our results in terms of 2D reprojection error in Table 1, 6D pose error in Table 2 and IoU metric in Table 4. We provide example pose predictions of our approach in Figure 3.

5.3.1 Comparative Accuracy

6D Accuracy in terms of projection error. In Table 1, we compare our results to those of Brachmann et al. [2] and to BB8 [26]. Both of these competing methods involve a multi-stage pipeline that comprises a 2D detection step followed by pose prediction and refinement. Since we do not have a refinement stage, we show in the table their results without and with it. In both cases, we achieve better 6D pose estimation accuracies.

In Table 4, we perform a similar comparison with SSD-6D [11], whose authors report their projection accuracy in terms of the IoU metric. That method also requires *a posteriori* refinement and our results are again better in both cases, even though SSD-6D relies on a large training set of rendered images that are sampled over a wide range of viewpoints and locations.

6D Accuracy in terms of the ADD metric. In Tables 2 and 3, we compare our methods against the other in terms of the average of the 3D distances, as described in Section 5.2. In Table 2, we give numbers before and after refinement for the competing methods. Before refinement, we outperform all the methods by a significant margin of at least 12%. After refinement, our pose estimates are still better than Brachmann et al. [2]. By assuming the additional knowledge of a full 3D CAD model and using it to further refine the pose, BB8 [1] and SSD-6D [2] boost their pose estimation accuracy.

Without any bells and whistles, our approach achieves state-of-the-art pose estimation accuracy in all the metrics without refinement. When compared against methods that rely on the additional knowledge of full 3D CAD models and pose refinement, it still achieves state-of-the-art performance in 2D projection error and IoU metrics and yields comparable accuracy in the ADD metric. Our approach could be used in conjunction with such refinement strategies to further increase the accuracy however this comes at a heavy computational cost as we describe below.

Method	w/o Refinement				w/ Refinement		
Object	Brachmann [2]	BB8 [26]	SSD-6D [11]	OURS	Brachmann [2]	BB8 [26]	SSD-6D [11]
Ape	-	***27.9***	0	21.62	33.2	40.4	**65**
Benchvise	-	62.0	0.18	***81.80***	64.8	**91.8**	80
Cam	-	***40.1***	0.41	36.57	38.4	55.7	**78**
Can	-	48.1	1.35	***68.80***	62.9	64.1	**86**
Cat	-	***45.2***	0.51	41.82	42.7	62.6	**70**
Driller	-	58.6	2.58	***63.51***	61.9	**74.4**	73
Duck	-	***32.8***	0	27.23	30.2	44.3	**66**
Eggbox	-	40.0	8.9	***69.58***	49.9	57.8	**100**
Glue	-	27.0	0	***80.02***	31.2	41.2	**100**
Holepuncher	-	42.4	0.30	***42.63***	52.8	**67.2**	49
Iron	-	67.0	8.86	***74.97***	80.0	**84.7**	78
Lamp	-	39.9	8.20	***71.11***	67.0	**76.5**	73
Phone	-	35.2	0.18	***47.74***	38.1	54.0	**79**
Average	32.3	43.6	2.42	***55.95***	50.2	62.7	**79**

Table 2. Comparison of our approach with state-of-the-art algorithms on LINEMOD in terms of ADD metric. We report percentages of correctly estimated poses.

Threshold	10%		30%		50%	
Object	[11]	OURS	[11]	OURS	[11]	OURS
Ape	0	**21.62**	5.62	**70.67**	19.95	**88.10**
Benchvise	0.18	**81.80**	2.07	**91.07**	10.62	**98.85**
Cam	0.41	**36.57**	34.52	**81.57**	63.54	**94.80**
Can	1.35	**68.80**	61.43	**99.02**	85.49	**99.90**
Cat	0.51	**41.82**	36.87	**90.62**	64.04	**98.80**
Driller	2.58	**63.51**	56.01	**99.01**	84.86	**99.80**
Duck	0	**27.23**	5.56	**70.70**	32.65	**89.39**
Eggbox	8.9	**69.58**	24.61	**81.31**	48.41	**98.31**
Glue	0	**80.02**	14.18	**89.00**	26.94	**97.20**
Holepuncher	0.30	**42.63**	18.23	**85.54**	38.75	**96.29**
Iron	8.86	**74.97**	59.26	**98.88**	88.31	**99.39**
Lamp	8.20	**71.11**	57.64	**98.85**	81.03	**99.62**
Phone	0.18	**47.74**	35.55	**91.07**	61.22	**98.85**
Average	2.42	**55.95**	31.65	**88.25**	54.29	**96.78**

Table 3. Comparison of our approach with SSD-6D [11] without refinement using different thresholds for the 6D pose metric.

[1] The authors do not report results without refinement, however they provided us with the accuracy numbers reported in Table 2.

[2] The authors were not able to provide their accuracy numbers without refinement for this metric, but made their code publicly available. We ran their code with provided pretrained models to obtain the 6D pose errors.

Method	w/o Refinement		w/ Refinement
Object	SSD-6D [11]	OURS	SSD-6D [11]
Ape	98.46	**99.81**	99
Benchvise	**100**	99.90	**100**
Cam	99.53	**100**	99
Can	**100**	99.81	**100**
Cat	99.34	**99.90**	99
Duck	99.04	**100**	98
Glue	97.24	**99.81**	98
Holepuncher	98.95	**99.90**	99
Iron	99.65	**100**	99
Lamp	99.38	**100**	99
Phone	99.91	**100**	100
Average	99.22	**99.92**	99.4
Driller	-	**100**	99
Eggbox	-	**99.91**	99

Table 4. Comparison of our approach against [11] on LINEMOD using IoU metric. The authors of [11] were able to provide us the results of our approach w/o the refinement.

5.3.2 Accuracy / Speed Trade-off

In Table 5, we report the computational efficiency of our approach for single object pose estimation in comparison to the state-of-the-art approaches [2, 11, 26]. Our approach runs at real-time performance in contrast to the existing approaches which fall short of it. In particular, our algorithm runs at least 5 times faster than the state-of-the-art techniques for single object pose estimation.

As can be seen in Table 2, pose refinement in Brachmann et al. increase the accuracy significantly by 17.9% at an additional run-time of 100 miliseconds per object. BB8 also gets a substantial improvement of 19.1% in accuracy at an additional run-time of 21 miliseconds per object. Even without correcting for the pose error, our approach outperforms Brachmann et al. and yields close accuracy to BB8 while being 16 times faster for single object pose estimation. As discussed also in [11], the unrefined poses computed from the bounding boxes of the SSD 2D object detector, are rather approximate. We confirmed this by running their publicly available code with the provided pretrained models. We report the accuracy numbers without the refinement using the ADD metric in Table 3 for different thresholds. While providing a good initialization for the subsequent pose processing, the pose estimates of SSD-6D without refinement are much less accurate than our approach. The further refinement increases the pose estimation accuracy significantly, however at the cost of a computational time of 24 miliseconds per object. Moreover, in contrast to our approach, the refinement requires the knowledge of the full 3D object CAD model.

In Figure 3, we show example results of our method on the LINEMOD. We include more visual results of our method in the supplementary material.

5.4. Multiple Object Pose Estimation

We use the OCCLUSION dataset to compare our approach to Brachmann et al. [2] for multi-object detection and report pose estimation accuracy as in [26]. The identity

Method	Overall Speed	Refinement runtime
Brachmann et al. [2]	2 fps	100 ms/object
Rad & Lepetit [26]	3 fps	21 ms/object
Kehl et al. [11]	10 fps	24 ms/object
OURS	50 fps	-

Table 5. Comparison of the overall computational runtime of our approach in comparison to [2, 11, 26]. We further provide the computational runtime induced by the pose refinement stage of [2, 11, 26]

of the objects cannot be assumed to be known *a priori* and has to be guessed. To this end, the method of [26] assumes that it has access to image crops based on the ground-truth 2D bounding boxes [3]. We make no such assumptions. Instead, we jointly detect the object in 2D, estimate its identity and predict its 6D pose. We generate our training images with the approach explained in Section 5.2. We further augment the LINEMOD training data by adding into the images objects extracted from other training sequences. We report our pose estimation accuracy in Figure 4 and demonstrate that even without assuming ground-truth information as in the case of [26], our method yields satisfactory pose accuracy in the case of severe occlusions. For object detection purposes, we consider an estimate to be correct if its detection IoU is larger than 0.5. Note that here the detection IoU corresponds to the overlap of the 2D bounding boxes of the object, rather than the overlap of the projected masks as is the case for the IoU metric defined in Sec 5.2. In Table 6, we report a mean average precision (MAP) of 0.48 which is similar to the accuracy reported by [2] and outperforms the ones reported by [8, 11].

Method	MAP
Hinterstoisser et al. [8]	0.21
Brachmann et al. [2]	0.51
Kehl et al. [11]	0.38
OURS	0.48

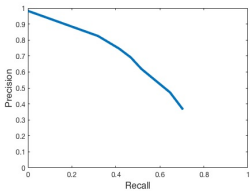


Table 6. The detection experiment on the Occlusion dataset [2]. (Left) Precision-recall plot. (Right)

Our approach provides accurate 6D poses with real-time performance. Upon one network invocation, our only computational overhead is an efficient PnP algorithm which operates on just 9 points per object. Furthermore we do not require full 3D colored object models to further refine our initial pose estimates. Our approach is therefore scalable to handle multiple objects as shown in Figure 5 and has only a negligible computational overhead of PnP (0.2 miliseconds/object) while the competing approaches [11] have a linear runtime growth.

We also evaluated the accuracy and speed of our ap-

[3] This it is not explicitly stated in [26], but the authors confirmed this to us in private email communication.

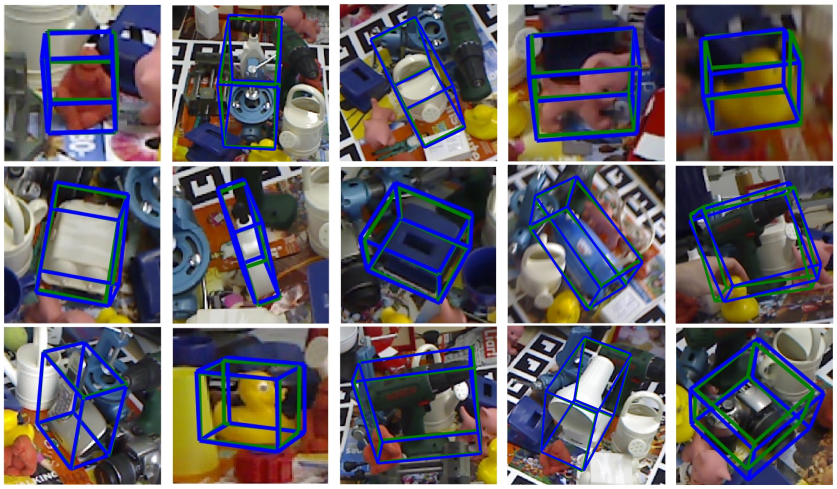


Figure 3. Pose estimation results of our approach. Note that our method can recover the 6D pose in these challenging scenarios, which involve significant amounts of clutter, occlusion and orientation ambiguity. In the last column, we show failure cases due to motion blur, severe occlusion and specularity (this figure is best viewed on a computer screen).

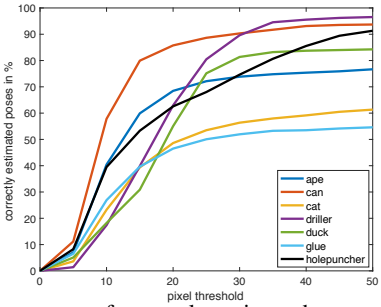


Figure 4. Percentage of correctly estimated poses as a function of the projection error for different objects of the Occlusion dataset [2].

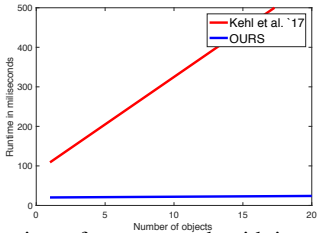


Figure 5. The runtime of our approach with increasing number of objects as compared to that of [11].

proach for different input resolutions. As explained in Section 3.1, we adopt a multi-scale training procedure and change the input resolution during training randomly as in [29]. This allows us to be able to change the input resolution at test-time and predict from images with higher resolution. This is especially useful for predicting the pose of small objects more robustly. As we do not have an initial step for 2D object detection and produce image crops which are then resized to higher resolutions for pose prediction as in [26], our approach requires better handling of the small objects. In Table 7, we compare the accuracy and computational efficiency of our approach for different input resolutions. With only 1% decrease in accuracy the average runtime per image is 94 ms and the runtime virtually remains the same for estimating the pose of multiple objects.

Resolution	2D projection metric	Speed
416×416	89.71	94 fps
480×480	90.00	67 fps
544×544	90.37	50 fps
688×688	90.65	43 fps

Table 7. Accuracy/speed trade-off of our method on the LINEMOD dataset. Accuracy reported is the percentage of correctly estimated poses w.r.t the 2D projection error. The same network model is used for all four input resolutions. Timings are on a Titan X (Pascal) GPU.

6. Conclusion

We have proposed a new CNN architecture for fast and accurate single-shot 6D pose prediction that naturally extends the single shot 2D object detection paradigm to 6D object detection. Our network predicts 2D locations of the projections of the objects 3D bounding box corners which involves predicting just a few more 2D points than for 2D bounding box regression. Given the predicted 2D corner projections, the 6D pose is computed via an efficient PnP method. For high accuracy, existing CNN-based 6D object detectors all refine their pose estimates during post-processing, a step that requires an accurate 3D object model and also incurs a runtime overhead per detected object. In contrast, our single shot predictions are very accurate which alleviates the need for refinement. Due to this, our method is not dependent on access to 3D object models and there is virtually no overhead when estimating the pose of multiple objects. Our method is real-time; it runs at 50 – 94 fps depending on the image resolution. This makes it substantially faster than existing methods.

Acknowledgements. This work was supported in part by the Swiss National Science Foundation. We would like to thank Mahdi Rad, Vincent Lepetit, Wadim Kehl, Fabian Manhardt and Slobodan Ilic for helpful discussions.

References

[1] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6D Object Pose Estimation Using 3D Object Coordinates. In *ECCV*, 2014. 1, 2, 5, 10

[2] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, et al. Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a Single RGB Image. In *CVPR*, 2016. 2, 5, 6, 7, 8, 10

[3] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Computational Geometry: Theory and Applications*, 43:601–610, 2010. 10

[4] C. Choi and H. I. Christensen. 3D Textureless Object Detection and Tracking: An Edge-Based Approach. In *IROS*, 2012. 1, 2

[5] C. Choi and H. I. Christensen. RGB-D Object Pose Estimation in Unstructured Environments. *Robotics and Autonomous Systems*, 2016. 1, 2

[6] A. Collet, M. Martinez, and S. S. Srinivasa. The MOPED Framework: Object Recognition and Pose Estimation for Manipulation. *The International Journal of Robotics Research*, 2011. 2

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) Challenge. *IJCV*, 2010. 6, 10

[8] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal Templates for Real-Time Detection of Texture-less Objects in Heavily Cluttered Scenes. In *ICCV*, 2011. 2, 7

[9] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model Based Training, Detection and Pose Estimation of Texture-less 3D Objects in Heavily Cluttered Scenes. In *ACCV*, 2012. 1, 2, 5, 6, 10

[10] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing Images using the Hausdorff Distance. *TPAMI*, 1993. 2

[11] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab. SSD-6D: Making RGB-Based 3D Detection and 6D Pose Estimation Great Again. In *ICCV*, 2017. 1, 2, 5, 6, 7, 8, 10

[12] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab. Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation. In *ECCV*, 2016. 1, 2

[13] A. Kendall, M. Grimes, and R. Cipolla. PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. In *ICCV*, 2015. 2

[14] K. Lai, L. Bo, X. Ren, and D. Fox. A Large-Scale Hierarchical Multi-View RGB-D Object Dataset. In *ICRA*, 2011. 1, 2

[15] K. Lai, L. Bo, X. Ren, and D. Fox. A Scalable Tree-Based Approach for Joint Object and Pose Recognition. In *AAAI*, 2011. 2

[16] V. Lepetit and P. Fua. Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Vision*, 2005. 2

[17] V. Lepetit, F. Moreno-Noguer, and P. Fua. EPnP: An Accurate O(n) Solution to the PnP problem. *IJCV*, 2009. 1, 2, 4

[18] Y. Li, L. Gu, and T. Kanade. Robustly Aligning a Shape Model and Its Application to Car Alignment of Unknown Pose. *TPAMI*, 2011. 2

[19] M.-Y. Liu, O. Tuzel, A. Veeraraghavan, and R. Chellappa. Fast Directional Chamfer Matching. In *CVPR*, 2010. 2

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. SSD: Single Shot MultiBox Detector. In *ECCV*, 2016. 1, 2

[21] D. G. Lowe. Fitting Parameterized Three-Dimensional Models to Images. *TPAMI*, 1991. 2

[22] D. G. Lowe. Object Recognition from Local Scale-Invariant Features. In *ICCV*, 1999. 1, 2

[23] S. Mahendran, H. Ali, and R. Vidal. 3D Pose Regression using Convolutional Neural Networks. *CVPRW*, 2017. 2

[24] F. Michel, A. Kirillov, E. Brachmann, A. Krull, S. Gumhold, B. Savchynskyy, and C. Rother. Global Hypothesis Generation for 6D Object Pose Estimation. In *CVPR*, 2017. 2

[25] P. Poirson, P. Ammirato, C.-Y. Fu, W. Liu, J. Kosecka, and A. C. Berg. Fast Single Shot Detection and Pose Estimation. In *3DV*, 2016. 2

[26] M. Rad and V. Lepetit. BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth. In *ICCV*, 2017. 1, 2, 5, 6, 7, 8, 10, 11, 12

[27] K. Ramnath, S. N. Sinha, R. Szeliski, and E. Hsiao. Car Make and Model Recognition using 3D Curve Alignment. In *WACV*, 2014. 2

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *CVPR*, 2016. 1, 2, 4

[29] J. Redmon and A. Farhadi. YOLO9000: Better, Faster, Stronger. *CVPR*, 2017. 1, 2, 4, 8

[30] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NIPS*, 2015. 2

[31] R. Rios-Cabrera and T. Tuytelaars. Discriminatively Trained Templates for 3d Object Detection: A Real Time Scalable Approach. In *ICCV*, 2013. 2

[32] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. 3D Object Modeling and Recognition using Local Affine-Invariant Image Descriptors and Multi-View Spatial Constraints. *IJCV*, 2006. 1, 2

[33] J. Sock, S. H. Kasaei, L. S. Lopes, and T.-K. Kim. Multi-view 6D Object Pose Estimation and Camera Motion Planning using RGBD Images. In *ICCV*, 2017. 2

[34] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. Render for CNN: Viewpoint Estimation in Images Using CNNs trained with Rendered 3D Model Views. In *ICCV*, 2015. 2

[35] S. Tulsiani and J. Malik. Viewpoints and Keypoints. In *CVPR*, 2015. 2

[36] L. Vacchetti, V. Lepetit, and P. Fua. Stable Real-Time 3D Tracking Using Online and Offline Information. *PAMI*, 26(10), 2004. 2

[37] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg. Pose Tracking from Natural Features on Mobile Phones. In *ISMAR*, 2008. 1, 2

[38] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox. PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes. *arXiv preprint arXiv:1711.00199*, 2017. 2

[39] C. Zach, A. Penate-Sanchez, and M.-T. Pham. A Dynamic Programming Approach for Fast and Robust Object Pose Recognition from Range Images. In *CVPR*, 2015. 2

[40] H. Zhang and Q. Cao. Combined Holistic and Local Patches for Recovering 6D Object Pose. In *ICCV*, 2017. 2

[41] M. Zhu, K. G. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. Phillips, M. Lecce, and K. Daniilidis. Single Image 3D Object Detection and Pose Estimation for Grasping. In *ICRA*, 2014. 2

Supplemental Material:
"Real-Time Seamless Single Shot 6D Object Pose Prediction"

In the supplemental material, we provide details on how the training images were prepared and on the proposed confidence function and the weighted prediction step. We also present qualitative results on OCCLUSION [1] and LINEMOD [9].

Training Images. As discussed in the main paper, we segment the foreground object in the images of the training set, using the segmentation masks provided and paste the segmented image over a random image as in [2, 11, 26]. Examples of such images, which are given as input to the network at training time are shown in Figure 6. This operation of removing the actual background prevents the network from overfitting to the background, which is similar for training and test images of LINEMOD. When we train a model without eliminating the background, in practice, we observe about 1% improvement in the 2D projection score.

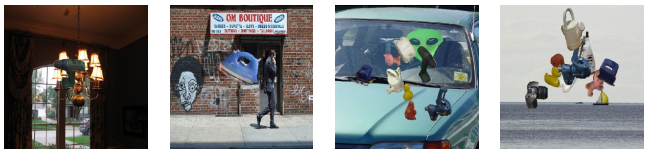
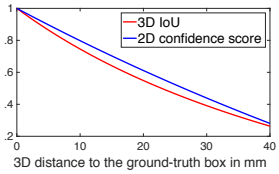


Figure 6. Using segmentation masks given in LINEMOD, we extract the foreground objects in our training images and composite them over random images from PASCAL VOC [7]. We also augment the training set by combining images of multiple objects taken from different training images.

Confidence function. We analyze in Figure 7 our confidence function in comparison to 3D cube IoU in terms of its value and runtime. We show that our confidence function closely approximates the actual 3D cube IoU while being much faster to compute.

Confidence-weighted prediction. In the final step of our method, we compute a weighted sum of multiple sets of predictions for the corners and the centroid, using associated confidence values as weights. On LINEMOD, this gave a 1–2% improvement in accuracy with the 2D projection metric. The first step involves scanning the full 17×17 grid to find the cell with the highest confidence for each potential object. We then consider a 3×3 neighborhood around it on the grid and prune the cells with confidence values



Method	Runtime per object (ms)
3D IoU	5.37
2D Conf. Score	0.18

Figure 7. Comparison of the 3D IoU and our 2D confidence score in value (Left) and runtime (Right). The model for the *Cam* object is shifted in x-dimension synthetically to produce a distorted prediction and projected on the image plane with randomly chosen 20 transformation matrices from LINEMOD. Scores are computed between the ground-truth references and distorted predictions. Results are averaged over all the trials. The runtime for 3D IoU is computed using the optimized PyGMO library that relies on [3].

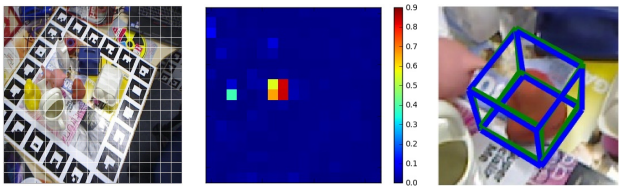


Figure 8. (Left) The 17×17 grid on a 544×544 image. (Middle) Confidence values for predictions of the *ape* object on the grid. (Right) Cropped view of our pose estimate (shown in blue) and the ground truth (shown in green). Here, three cells next to the best cell have good predictions and their combination gives a more accurate pose than the best prediction alone (best viewed in color).

lower than the detection threshold of 0.5. On the remaining cells, we compute a confidence-weighted average of the associated predicted 18-dimensional vectors, where the eight corner points and the centroid have been stacked to form the vector. The averaged coordinates are then used in the PnP method. This *s*ub-pixel refinement on the grid usually improves the pose of somewhat large objects that occupy several adjoining cells in the grid. Figure 8 shows an example where the *ape* object lies between two adjoining cells and the confidence weighting improves the pose accuracy.

Qualitative Results. We show qualitative results from the OCCLUSION [1] and LINEMOD [9] datasets in Figures 9 to 14. These examples show that our method is robust to severe occlusions, rotational ambiguities in appearance, reflections, viewpoint change and scene clutter.

Figure 9. Results on the OCCLUSION dataset. Our method is quite robust against severe occlusions in the presence of scene clutter and rotational pose ambiguity for symmetric objects. (left) Input images, (middle) 6D pose predictions of multiple objects, (right) A magnified view of the individual 6D pose estimates of six different objects is shown for clarity. In each case, the 3D bounding box is rendered on the input image. The following color coding is used – APE (gold), BENCHVISE (green), CAN (red), CAT (purple), DRILLER (cyan), DUCK (black), GLUE (orange), HOLEPUNCHER (blue). In addition to the objects from the OCCLUSION dataset, we also visualize the pose predictions of the *Benchvise* object from the LINEMOD dataset. As in [26], we do not evaluate on the *Eggbox* object, as more than 70% of close poses are not seen in the training sequence. This image is best viewed on a computer screen.

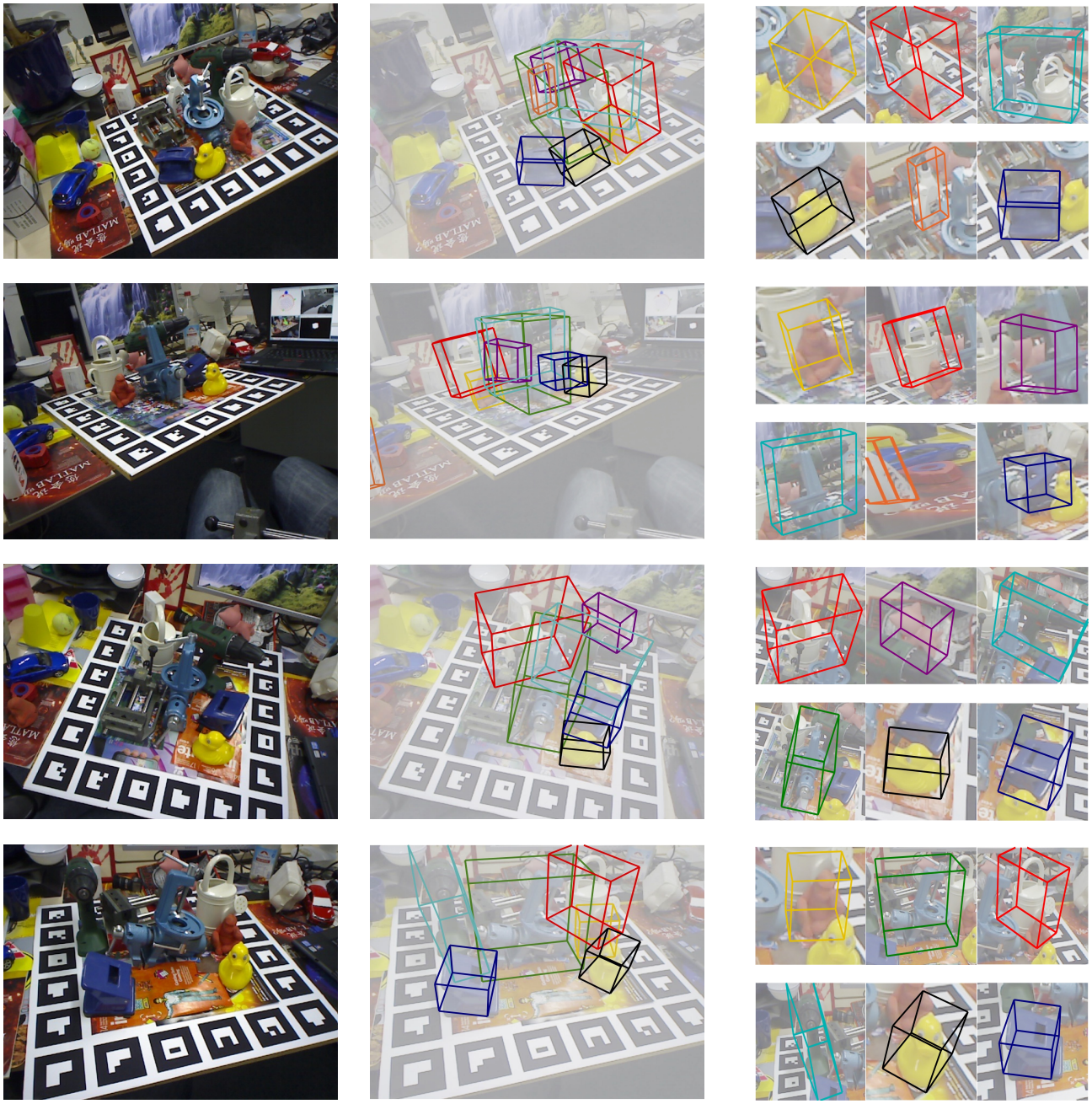


Figure 10. Results on the OCCLUSION dataset. Our method is quite robust against severe occlusions in the presence of scene clutter and rotational pose ambiguity for symmetric objects. (left) Input images, (middle) 6D pose predictions of multiple objects, (right) A magnified view of the individual 6D pose estimates of six different objects is shown for clarity. In each case, the 3D bounding box is rendered on the input image. The following color coding is used – APE (gold), BENCHVISE (green), CAN (red), CAT (purple), DRILLER (cyan), DUCK (black), GLUE (orange), HOLEPUNCHER (blue). In addition to the objects from the OCCLUSION dataset, we also visualize the pose predictions of the *Benchvise* object from the LINEMOD dataset. As in [26], we do not evaluate on the *Eggbox* object, as more than 70% of close poses are not seen in the training sequence. This image is best viewed on a computer screen.

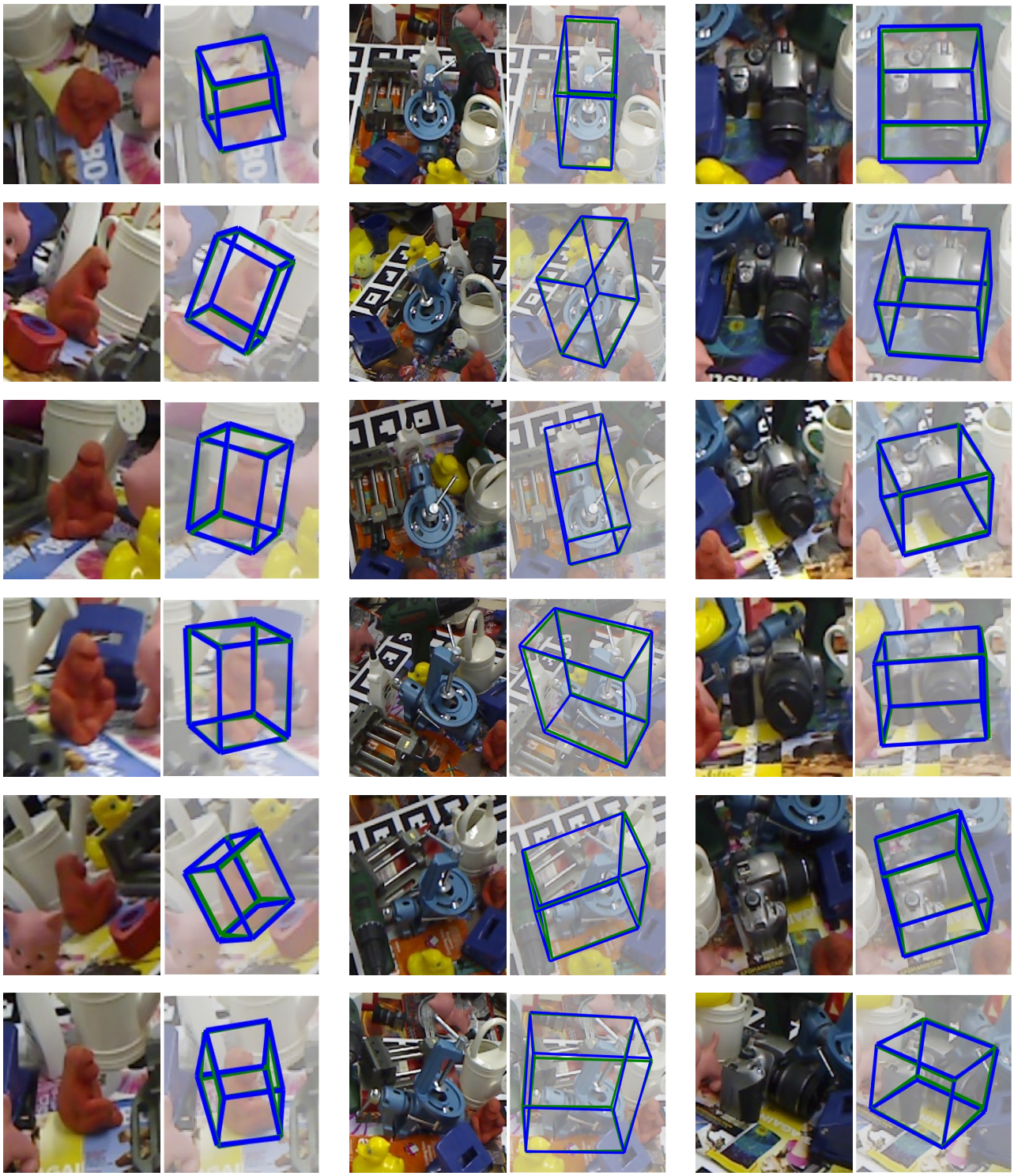


Figure 11. Example results on the LINEMOD dataset: (left) APE, (middle) BENCHVISE, (right) CAM. The projected 3D bounding boxes are rendered over the image and they have been cropped and resized for ease of visualization. The blue cuboid is rendered using our pose estimate whereas the green cuboid is rendered using the ground truth object pose. Note that the input image dimension is 640×480 pixels and the objects are often quite small. Noticeable scene clutter and occlusion makes these examples challenging.

Figure 12. Example results on the LINEMOD dataset: (left) CAN, (middle) CAT, (right) DRILLER. The projected 3D bounding boxes are rendered over the image and they have been cropped and resized for ease of visualization. The blue cuboid is rendered using our pose estimate whereas the green cuboid is rendered using the ground truth object pose. Note that the input image dimension is 640×480 pixels and the objects are often quite small. Noticeable scene clutter and occlusion makes these examples challenging.

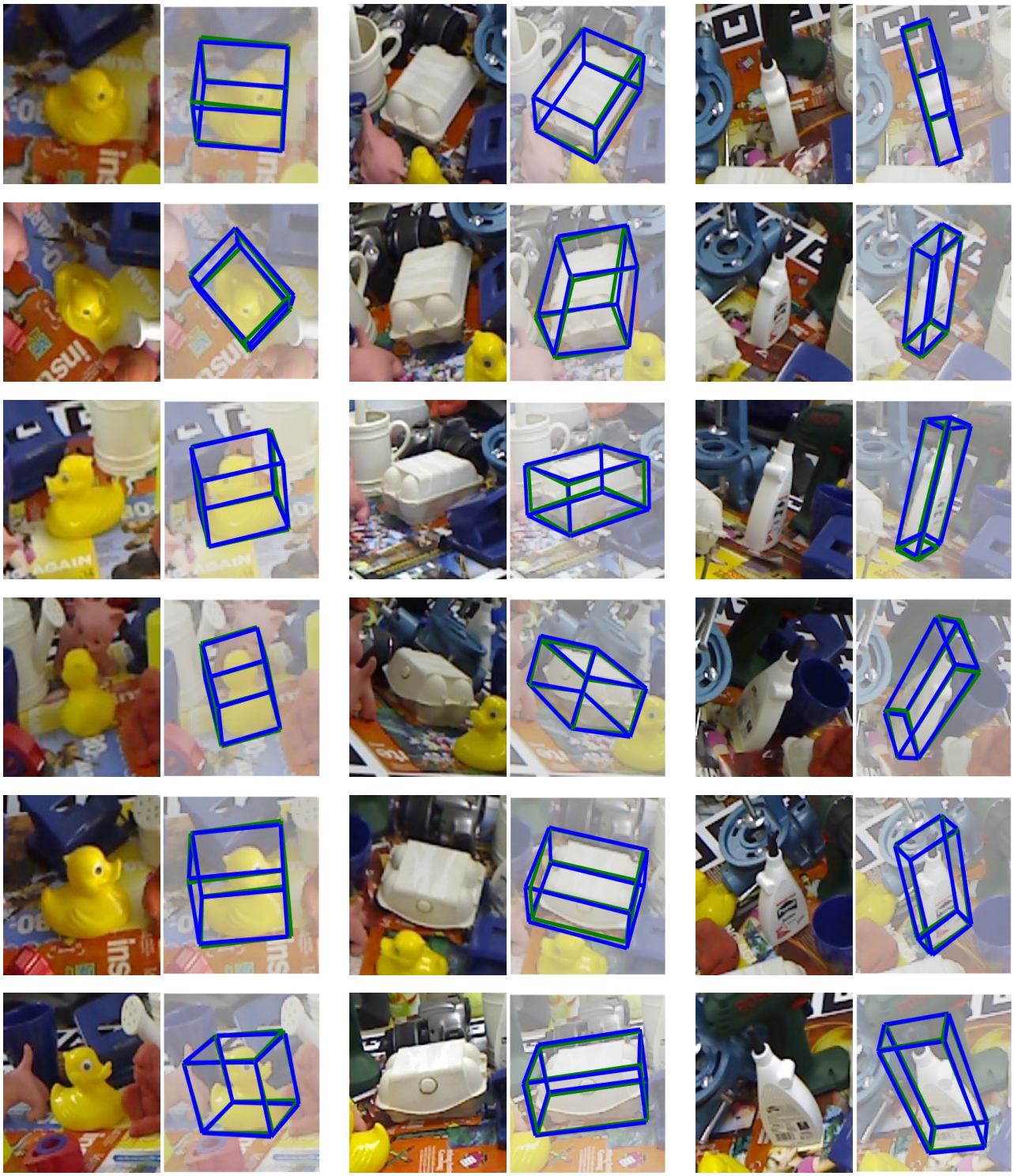


Figure 13. Example results on the LINEMOD dataset: (left) DUCK, (middle) EGGBOX, (right) GLUE. The projected 3D bounding boxes are rendered over the image and they have been cropped and resized for ease of visualization. The blue cuboid is rendered using our pose estimate whereas the green cuboid is rendered using the ground truth object pose. Note that the input image dimension is 640×480 pixels and the objects are often quite small. Noticeable scene clutter and occlusion makes these examples challenging.

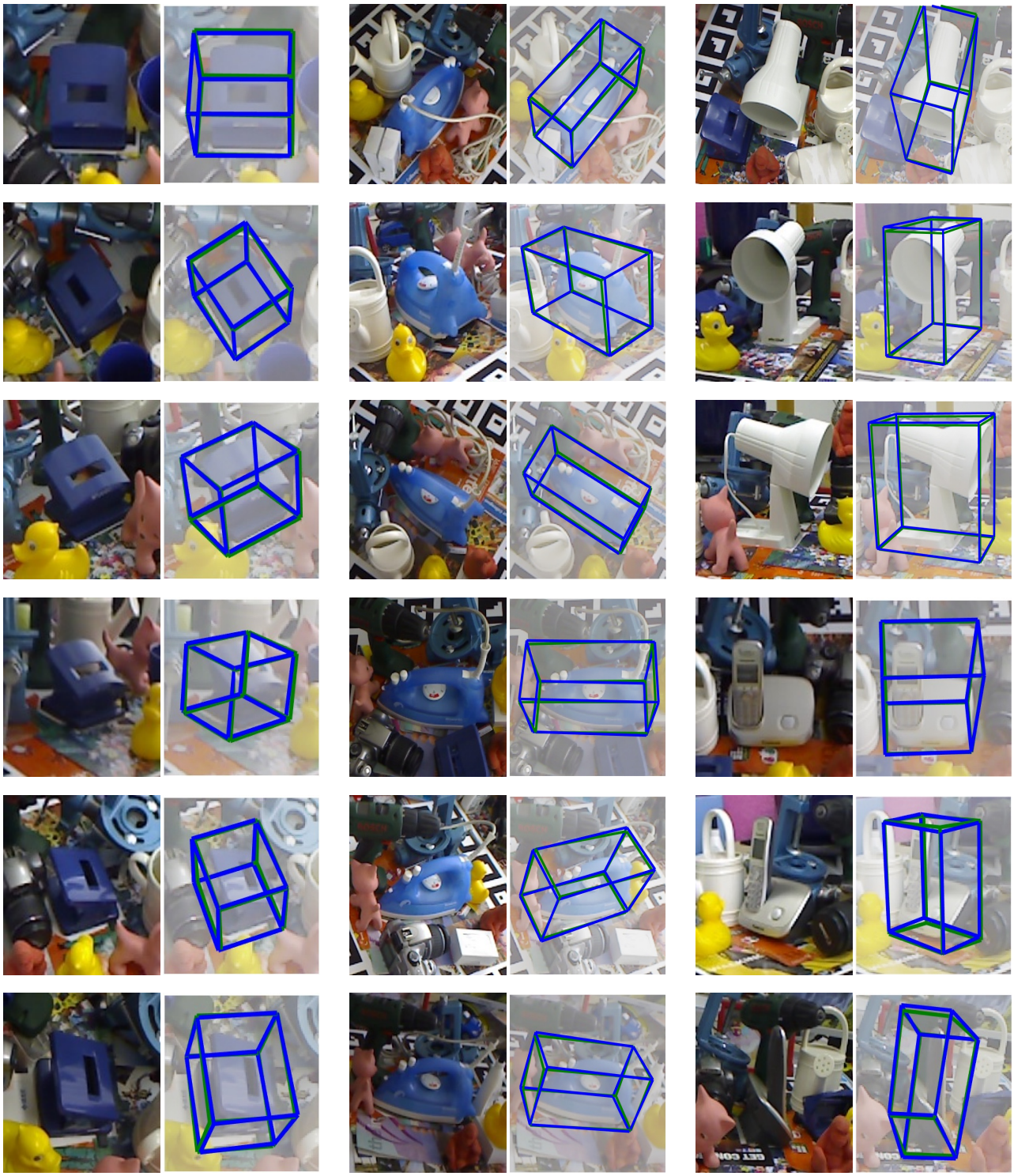


Figure 14. Example results on the LINEMOD dataset: (left) HOLEPUNCHER, (middle) IRON, (right) LAMP and PHONE. The projected 3D bounding boxes are rendered over the image and they have been cropped and resized for ease of visualization. The blue cuboid is rendered using our pose estimate whereas the green cuboid is rendered using the ground truth object pose. Note that the input image dimension is 640×480 pixels and the objects are often quite small. Noticeable scene clutter and occlusion makes these examples challenging.