International Journal of Computer Vision
https://doi.org/10.1007/s11263-019-01250-9

# DeepIM: Deep Iterative Matching for 6D Pose Estimation

Yi Li[1,2] · Gu Wang[2] · Xiangyang Ji[2] · Yu Xiang[3] · Dieter Fox[1,3]

Received: 9 February 2019 / Accepted: 19 September 2019

**Abstract**
Estimating 6D poses of objects from images is an important problem in various applications such as robot manipulation and virtual reality. While direct regression of images to object poses has limited accuracy, matching rendered images of an object against the input image can produce accurate results. In this work, we propose a novel deep neural network for 6D pose matching named DeepIM. Given an initial pose estimation, our network is able to iteratively refine the pose by matching the rendered image against the observed image. The network is trained to predict a relative pose transformation using a disentangled representation of 3D location and 3D orientation and an iterative training process. Experiments on two commonly used benchmarks for 6D pose estimation demonstrate that DeepIM achieves large improvements over state-of-the-art methods. We furthermore show that DeepIM is able to match previously unseen objects.

**Keywords** 3D object recognition · 6D object pose estimation · Object tracking

## 1 Introduction

Localizing objects in 3D from images is important in many real world applications. For instance, in a robot manipulation task, the ability to recognize the 6D pose of objects, i.e., 3D location and 3D orientation of objects, provides useful information for grasp and motion planning. In a virtual reality application, 6D object pose estimation enables virtual interactions between human and objects. While several recent techniques have used depth cameras for object pose estimation, such cameras have limitations with respect to frame rate, field of view, resolution, and depth range, making it very difficult to detect small, thin, transparent, or fast moving objects. Unfortunately, RGB-only 6D object pose estimation is still a challenging problem, since the appearance of objects in the images changes according to a number of factors, such as lighting, pose variations, and occlusions between objects. Furthermore, a robust 6D pose estimation method needs to handle both textured and textureless objects.

Traditionally, the 6D pose estimation problem has been tackled by matching local features extracted from an image to features in a 3D model of the object (Lowe 1999; Rothganger et al. 2006; Collet et al. 2011). By using the 2D–3D correspondences, the 6D pose of the object can be recovered. Unfortunately, such methods cannot handle textureless objects well since only few local features can be extracted for them. To handle textureless objects, two classes of approaches were proposed in the literature. Methods in the first class learn to estimate the 3D model coordinates of pixels or keypoints of the object in the input image. In this way, the 2D–3D correspondences are established for 6D pose estimation (Brachmann et al. 2014; Rad and Lepetit 2017; Tekin et al. 2017). Methods in the second class convert the 6D pose estimation problem into a pose classification problem by discretizing the pose space (Hinterstoisser et al. 2012b) or into a pose regression problem (Xiang et al. 2018). These methods can deal with textureless objects, but they are not able to achieve highly accurate pose estimation, since small errors

Communicated by Cristian Sminchisescu.

✉ Yi Li
yili.matrix@gmail.com

Gu Wang
wangg16@mails.tsinghua.edu.cn

Xiangyang Ji
xyji@tsinghua.edu.cn

Yu Xiang
yux@nvidia.com

Dieter Fox
dieterf@nvidia.com

[1] University of Washington, Seattle, USA
[2] Tsinghua University and BNRist, Beijing, China
[3] NVIDIA, Seattle, WA, USA

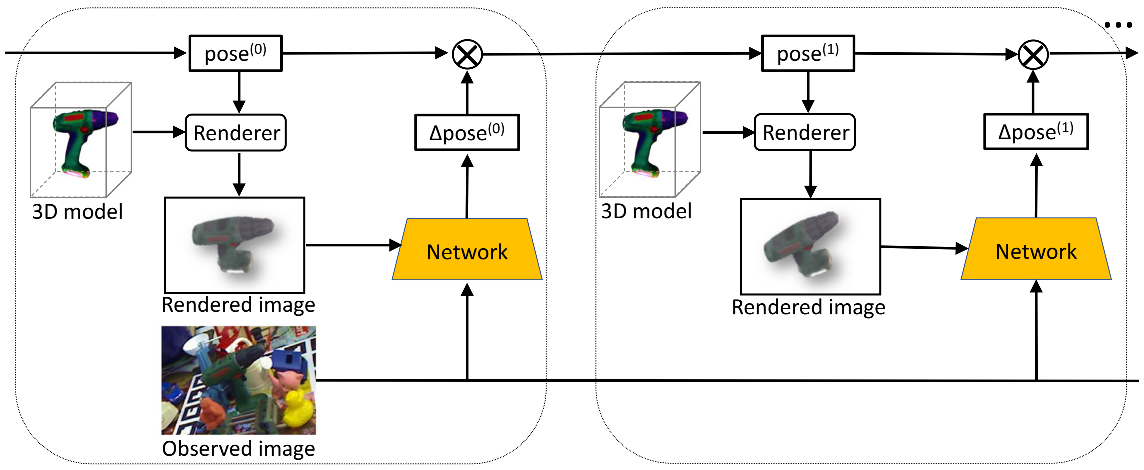


**Fig. 1** We propose DeepIM, a deep iterative matching network for 6D object pose estimation. The network is trained to predict a relative SE(3) transformation that can be applied to an initial pose estimation for iterative pose refinement. Given a 6D pose estimation of an object, which can be the output of other pose estimation methods like PoseCNN (Xiang et al. 2018) (pose$^{(0)}$ in the figure) or the refined pose from previous iteration (pose$^{(1)}$ in the figure), along with the 3D model of the object, we generate the rendered image showing the appearance of the target object under this rough pose estimation. With the image pairs of rendered image and observed image, the network predicts a relative transformation (1pose in the figure) which can be applied to refine the input pose. The refined pose can be used as the input pose of next iteration and therefore the process can be repeated until the refined pose converges or the number of iterations reaches a pre-determined number

in the classification or regression stage directly lead to pose mismatches. A common way to improve the pose accuracy is pose refinement: Given an initial pose estimation, a synthetic RGB image can be rendered and used to match against the target input image. Then a new pose is computed to increase the matching score. Existing methods for pose refinement use either hand-crafted image features (Tjaden et al. 2017) or matching score functions (Rad and Lepetit 2017).

In this work, we propose DeepIM, a new refinement technique based on a deep neural network for iterative 6D pose matching. Given an initial 6D pose estimation of an object in a test image, DeepIM predicts a relative SE(3) transformation that matches a rendered view of the object against the observed image, or in other words, it predicts the relative rotation and translation that can refine the initial 6D pose estimation. By iteratively re-rendering the object based on the improved pose estimates, the two input images to the network become more and more similar, thereby enabling the network to generate more and more accurate pose estimates. Figure 1 illustrates the iterative matching procedure of our network for pose refinement.

This work makes the following main contributions. (1) We introduce a deep network for iterative, image-based pose refinement that does not require any hand-crafted image features and automatically learns an internal refinement mechanism. (2) We propose a disentangled representation of the SE(3) transformation between object poses to achieve accurate pose estimates. This representation also enables our approach to refine pose estimates of unseen objects. (3) We have conducted extensive experiments on the LINEMOD (Hinterstoisser et al. 2012b) and the Occlusion LINEMOD (Brachmann et al. 2014) datasets to evaluate the accuracy and various properties of DeepIM. These experiments show that our approach achieves large improvements over state-of-the-art RGB-only methods on both datasets. Furthermore, initial experiments demonstrate that DeepIM is able to accurately match poses for textureless objects [T-LESS (Hodan et al. 2017)] and for unseen objects (Wu et al. 2015). The rest of the paper is organized as follows. After reviewing related works in Sect. 2, we describe our approach for pose matching in Sect. 3. Experiments are presented in Sect. 4, and Sect. 5 concludes the paper.

## 2 Related Work

We review representative works on 6D pose estimation in the literature.

### 2.1 RGB Based 6D Pose Estimation

Traditionally, object pose estimation using RGB images is tackled by matching local features (Lowe 1999; Rothganger et al. 2006; Collet et al. 2011). In this paradigm, a 3D model of an object is first reconstructed and local features of the object are attached to the 3D model. Keypoint-based features such as SIFT (Lowe 1999) or SURF (Bay et al. 2008) are widely used. Given an input image, local features extracted from the image are matched against features on the 3D model. By filtering out incorrect matches using robust estimation techniques such as RANSAC (Nistér 2005), the 6D pose of the object can be recovered using the 2D-to-3D correspondences between the local features. Local-feature matching based methods can handle partial occlusions between objects as long as the features on the visual part of the object are sufficient to determine the 6D pose. However, these methods cannot handle textureless objects well, since rich texture on the object is required in order to detect these features robustly.

In contrast, template-matching based methods are capable of handling textureless objects (Jurie and Dhome 2001; Liu et al. 2010; Gu and Ren 2010; Hinterstoisser et al. 2012a). In this paradigm, templates of an object are first constructed, where examples of templates are renderings of the object from the 3D object model or Histogram of Oriented Gradients (HOG) (Dalal and Triggs 2005) templates from different viewpoints. Then these templates are matched against the input image to determine the location and orientation of the target object in the input image. The drawback of template-matching based methods is that they are not robust to occlusions between objects. When the target object is heavily occluded, the matching score is usually low which may result in incorrect pose estimation.

Recent approaches apply machine learning, especially deep learning, for 6D pose estimation using RGB images (Brachmann et al. 2014; Krull et al. 2015). Learning techniques are employed to detect object keypoints for matching or learn better feature representations for pose estimation. The state-of-the-art methods (Rad and Lepetit 2017; Kehl et al. 2017; Tekin et al. 2017; Xiang et al. 2018; Tremblay et al. 2018) augment deep learning based object detection or segmentation methods (Girshick 2015; Long et al. 2015; Liu et al. 2016; Redmon et al. 2016) for 6D pose estimation. For example, Rad and Lepetit (2017), Tjaden et al. (2017) and Tremblay et al. (2018) utilize deep neural networks to detect keypoints on the objects, and then compute the 6D pose by solving the PnP problem. Kehl et al. (2017) and Xiang et al. (2018) employ deep neural networks to detect objects in the input image, and then classify or regress the detected object to its pose. A recent work Sundermeyer et al. (2018) uses an autoencoder to map the object in the image to a vector and search for the most similar vector in a pre-generated codebook for pose estimation. Overall, learning-based methods achieve better performance than traditional methods, largely due to the ability of learning a powerful feature representation for pose estimation.

### 2.2 Depth Based 6D Pose Estimation

From another point of view, the 6D pose estimation problem can be tackled using depth images. Given a 3D model of an object and an input depth image, the problem is formulated as aligning the two point clouds computed from the 3D model and the depth image, respectively, which is also known as the geometric registration problem. Roughly speaking, geometric registration methods can be classified as local refinement methods and global registration methods. The most well-known local refinement algorithm is the Iterative Closest Point (ICP) algorithm (Besl and McKay 1992) and its variants (Rusinkiewicz and Levoy 2001; Salvi et al. 2007; Tam et al. 2013). Given an initial pose estimation, the ICP algorithm iterates between finding the correspondences between points and refining the pose estimation using the new correspondences. In general, local refinement algorithms are sensitive to the initial pose. If the initial pose estimation is not close enough, the algorithm may converge to a local mimimum.

Global registration methods (Mellado et al. 2014; Theiler et al. 2015; Zhou et al. 2016; Yang et al. 2016) solve a more challenging problem by not assuming an initial pose estimate. A common strategy is to utilize iterative model fitting frameworks such as RANSAC. In each iteration, a set of point correspondences are sampled, and an alignment is computed and evaluated using the sampled correspondences. The limitation of most global registration methods is that they are computationally expensive. Also, the registration quality heavily depends on the quality of the 3D model and the scanned point cloud. In order to improve the registration performance, features on point clouds are also introduced for matching. These include point pairs (Mian et al. 2006; Hinterstoisser et al. 2016), spin-images (Johnson and Hebert 1999), and point-pair histograms (Rusu et al. 2009; Tombari et al. 2010). Similar to the trend in image-based matching, recent approaches Wang et al. (2019) propose to learn point features for registration, such as applying deep neural networks to point clouds (Qi et al. 2017).

### 2.3 RGB-D Based 6D Pose Estimation

When both RGB images and depth images are available, they can be combined to improve 6D pose estimation. A common strategy is to estimate an initial pose of an object based on the color image, and then refine the pose using depth-based local refinement algorithms such as ICP (Hinterstoisser et al. 2012b; Michel et al. 2017; Zeng et al. 2017).

For example, Hinterstoisser et al. (2012b) renders the 3D model of an object into templates of color images, and then matches these templates against the input image to estimate an initial pose. The final pose estimation is obtained via ICP refinement on the initial pose. Brachmann et al. (2014), Brachmann et al. (2016) and Michel et al. (2017) regress each pixel on the object in the input image to the 3D coordinate of that pixel on the 3D model. When depth images are available, the 3D coordinate regression establishes correspondences between 3D scene points and 3D model points, from which the 6D pose can be computed by solving a least-

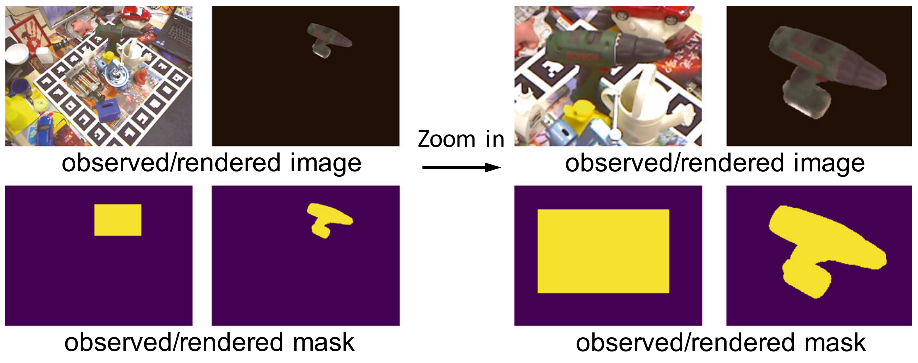


**Fig. 2** DeepIM operates on a zoomed in, up-sampled input image, the rendered image, and the two object masks (480 × 640 in our case after zooming in). More specifically, we enlarge the bounding box of the object in the rendered image, crop the corresponding patch using the enlarged bounding box in both image pairs and mask pairs and then up-sample them to high resolution. Notice that the aspect ratio is kept during this process to avoid image distortion. See Sect. 3.1 for more details

squares problem. PoseCNN (Xiang et al. 2018) introduces an end-to-end neural network for 6D object pose estimation using RGB images only. Given an initial pose from the network, a customized ICP method is applied to refine the pose. A recent work Wang et al. (2019) introduces a neural network that combines RGB images and depth images for 6D pose estimation, and an iterative pose refinement network using point clouds as input.

### 2.4 RGB Versus RGB-D

Overall, the performance of RGB-based methods is still not comparable to that of the RGB-D based methods. We believe that this performance gap is largely due to the lack of an effective pose refinement procedure using RGB images only. Manhardt et al. (2018) which is published at the same time as ours introduces a method to refine 6D object poses with only RGB images, but there is still a large performance gap between Manhardt et al. (2018) and depth-based methods. Our work is complementary to existing 6D pose estimation methods by providing a novel iterative pose matching network for pose refinement on RGB images.

The approaches most related to ours are the object pose refinement network in Rad and Lepetit (2017) and the iterative hand pose estimation approaches in Carreira et al. (2016) and Oberweger et al. (2015). Compared to these techniques, our network is designed to directly regress to relative SE(3) transformations. We are able to do this due to our disentangled representation of rotation and translation and the reference frame we used for rotation, which also allows our approach to match unseen objects. As shown in Mousavian et al. (2017), the choice of reference frame is important to achieve good pose estimation results. Our work is also related to recent visual servoing methods based on deep neural networks (Saxena et al. 2017; Costante and Ciarfuglia 2018) that estimate the relative camera pose between two image frames, while we focus on 6D pose refinement of objects. Recent works Garon et al. (2016) and Garon and Lalonde (2017) that focus on tracking could predict the transformation of the object pose between previous frame and current frame and have the potential to be used for pose refinement.

## 3 DeepIM Framework

In this section, we describe our deep iterative matching network for 6D pose estimation. Given an observed image and an initial pose estimate of an object in the image, we design the network to directly output a relative SE(3) transformation that can be applied to the initial pose to improve the estimate. We first present our strategy of zooming in the observed image and the rendered image that are used as inputs of the network. Then we describe our network architecture for pose matching. After that, we introduce a disentangled representation of the relative SE(3) transformation and a new loss function for pose regression. Finally, we describe our procedure for training and testing the network.

### 3.1 High-Resolution Zoom In

It can be difficult to extract useful features for matching if objects in the input image are very small. To obtain enough details for pose matching, we zoom in the observed image and the rendered image before feeding them into the network, as shown in Fig. 2. Specifically, in the $i$-th stage of the iterative matching, given a 6D pose estimate $\mathbf{p}^{(i-1)}$ from the previous step, we render a synthetic image using the 3D object model viewed according to $\mathbf{p}^{(i-1)}$.

We additionally generate one foreground mask for the observed image and rendered image. The four images are cropped using an enlarged bounding box according to the observed mask and the rendered mask, where we make sure

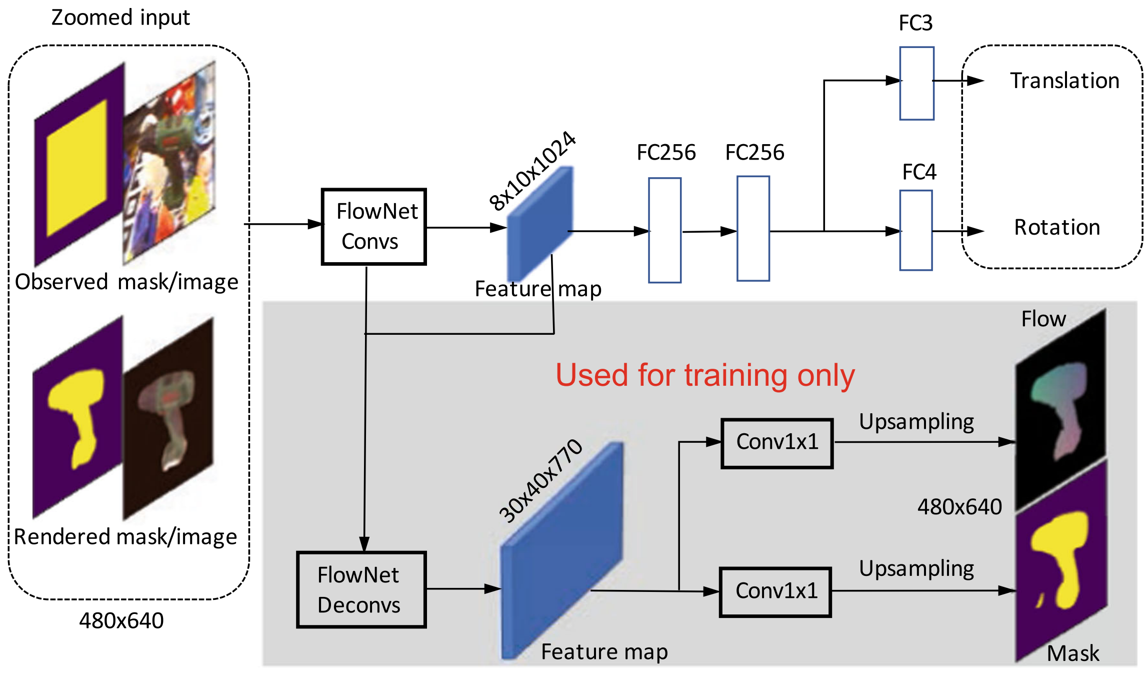


**Fig. 3** DeepIM uses a FlowNetSimple backbone to predict a relative SE(3) transformation to match the observed and rendered image of an object. Taking observed image and rendered image and their corresponding masks as input, the convolution layers output a feature map which then be forwarded through several fully connected layers to predict the translation and rotation. The same feature map, combined with feature maps in the previous layers, will also be used to predict flow and foreground mask during training

the enlarged bounding box has the same aspect ratio as the input image and is centered at the 2D projection of the origin of the 3D object model.

In more detail, given the rendered mask $\mathbf{m}_{\text{rend}}$ and the observed mask $\mathbf{m}_{\text{obs}}$, the cropping patch is computed as

$$\begin{aligned} x_{\text{dist}} &= \max(|l_{\text{obs}} - x_c|, |l_{\text{rend}} - x_c|, \\ &\qquad |r_{\text{obs}} - x_c|, |r_{\text{rend}} - x_c|), \\ y_{\text{dist}} &= \max(|u_{\text{obs}} - y_c|, |u_{\text{rend}} - y_c|, \\ &\qquad |d_{\text{obs}} - y_c|, |d_{\text{rend}} - y_c|), \\ \text{width} &= \max(x_{\text{dist}}, y_{\text{dist}} \cdot r) \cdot 2\lambda, \\ \text{height} &= \max(x_{\text{dist}}/r, y_{\text{dist}}) \cdot 2\lambda, \end{aligned} \tag{1}$$

where $u_*, d_*, l_*, r_*$ denotes the upper, lower, left, right bound of foreground mask of observed or rendered images, $x_c, y_c$ represent the 2D projection of the center of the object in $\mathbf{img}_{\text{rend}}$, $r$ represent the aspect ratio of the origin image (width/height), $\lambda$ denotes the expand ratio, which is fixed to 1.4 in the experiment in order to make the expanded patch is roughly twice than the nested one. Then this patch is bilinearly sampled to the size of the original image, which is $480 \times 640$ in this paper. By doing so, not only the object is zoomed in without being distorted, but also the network is provided with the information about where the center of the object lies.

## 3.2 Network Structure

Figure 3 illustrates the network architecture of DeepIM. The observed image, the rendered image, and the two masks, are concatenated into an eight-channel tensor input to the network (3 channels for observed/rendered image, 1 channel for each mask). We use the FlowNetSimple architecture from Dosovitskiy et al. (2015) as the backbone network, which is trained to predict optical flow between two images. We tried using the VGG16 image classification network (Simonyan and Zisserman 2014) as the backbone network, but the results were very poor, confirming the intuition that a representation related to optical flow is very useful for pose matching (Wang et al. 2017).

The pose estimation branch takes the feature map after 10 convolution layers from FlowNetSimple as input. It contains two fully-connected layers each with dimension 256, followed by two additional fully-connected layers for predicting the quaternion of the 3D rotation and the 3D translation, respectively.

During training, we also add two auxiliary branches to regularize the feature representation of the network and increase training stability and performance, see Sect. 4.4 and Table 1 for more details. One branch is trained for predicting optical flow between the rendered image and the observed image,

**Table 1** Ablation study on the role of mask prediction and flow prediction branch

| Methods | | 5 cm 5° | 6D Pose | Proj. 2D |
|---|---|---|---|---|
| Mask | Flow | | | |
| ✓ | ✓ | 93.9 ± 0.7 | 82.5 ± 1.7 | 98.2 ± 0.3 |
| ✓ | | 91.7 ± 0.4 | 82.5 ± 1.6 | 97.7 ± 0.1 |
| | ✓ | 89.2 ± 2.1 | 63.7 ± 3.4 | 98.4 ± 0.2 |
| | | 89.6 ± 0.8 | 72.3 ± 1.1 | 98.1 ± 0.1 |

The networks are trained 5 times for each setting on the object ape of the LINEMOD dataset. The numbers denote mean ± standard deviation

and the other branch for predicting the foreground mask of the object in the observed image.

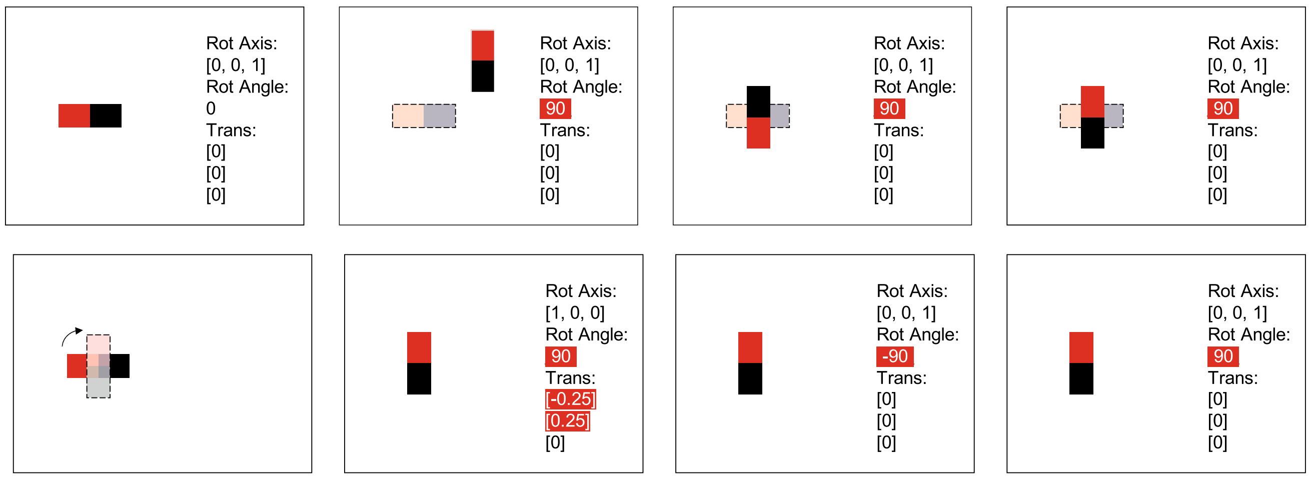
## 3.3 Disentangled Transformation Representation

The representation of the coordinate frames and the relative SE(3) transformation **1p** between the current pose estimate and the target pose has important ramifications for the performance of the network. Ideally, we would like (1) the individual components of these transformations to be maximally dis-entangled, thereby not requiring the network to learn unnecessarily complex geometric relationships between translations and rotations, and (2) the transformations to be independent of the intrinsic camera parameters and the actual size and coordinate system of an object, thereby enabling the network to reason about changes in object appearance rather than accurate distance estimates.

The most obvious choice are camera coordinates to represent object poses and transformations. Denote the relative rotation and translation as $[\mathbf{R}_{\mathbf{1}}|\mathbf{t}_{\triangle}]$ (We denote $\mathbf{R}_{*}$ as rotation and and $\mathbf{t}_{*}$ as translation in this paper). Given a source object pose $[\mathbf{R}_{\text{src}}|\mathbf{t}_{\text{src}}]$, the transformed target pose would be as follows:

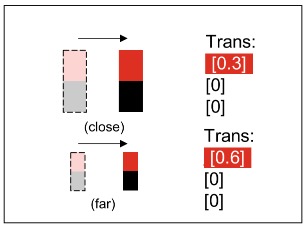
$$\mathbf{R}_{\text{tgt}} = \mathbf{R}_1 \mathbf{R}_{\text{src}}, \quad \mathbf{t}_{\text{tgt}} = \mathbf{R}_1 \mathbf{t}_{\text{src}} + \mathbf{t}_1, \tag{2}$$

where $[\mathbf{R}_{\text{tgt}}|\mathbf{t}_{\text{tgt}}]$ denotes the target pose resulting from the transformation. The $\mathbf{R}_1 \mathbf{t}_{\text{src}}$ term indicates that a rotation will cause the object not only to rotate, but also translate in the image even if the translation vector $\mathbf{t}_1$ equals to zero. Column (b) in Fig. 4 illustrates this connection for an object rotating in the image plane. In standard camera coordinates, the translation $\mathbf{t}_1$ of an object is in the 3D metric space (meter, for instance), which couples object size with distance in the metric space. This would require the network to memorize the actual size of each object in order to transform mis-matches in images to distance offsets. It is obvious that such a representation is not appropriate, particularly for matching unknown objects.

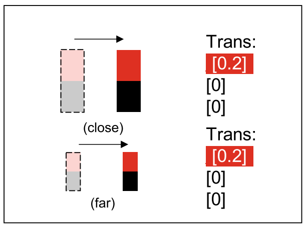

Rot Axis: [0, 0, 1] Rot Angle: 0 Trans: [0] [0] [0]
Rot Axis: [0, 0, 1] Rot Angle: 90 Trans: [0] [0] [0]
Rot Axis: [0, 0, 1] Rot Angle: 90 Trans: [0] [0] [0]
Rot Axis: [0, 0, 1] Rot Angle: 90 Trans: [0] [0] [0]
Rot Axis: [1, 0, 0] Rot Angle: 90 Trans: [-0.25] [0.25] [0]
Rot Axis: [0, 0, 1] Rot Angle: -90 Trans: [0] [0] [0]
Rot Axis: [0, 0, 1] Rot Angle: 90 Trans: [0] [0] [0]

**(a)** Initial pose **(b)** Camera coord. **(c)** Model coord. **(d)** disentangled coord.

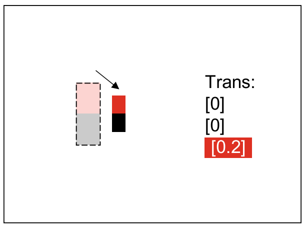
**Fig. 4** Rotations using different coordinate systems. (Upper row) The panels show how a 90° rotation in the image plane axis changes the position of the object shown in (**a**). In the camera coordinate system, the center of rotation is in the center of the image, thereby causing an undesired translation in addition to the object rotation. In the model coordinate frame, as the frame of the object model can be defined arbitrarily, an object might rotate along any axis given the same rotation vector. Shown here is a CCW rotation, but the same axis might also result in an out of plane rotation for a differently defined object coordinate frame. In our disentangled representation, the center of rotation is in the center of the object and the axes are defined parallel to the camera axes. As a result, a rotation around a specific axis always results in the same object rotation, independent of the object. (Lower row) Rotation vectors a network would have to predict in order to achieve an in-place rotation using the different coordinate systems. Notice the extra translations required to compensate for the translation caused by the rotation using camera coordinates (**b**). In model coordinates, the network would have to learn the frame specified for the object model in order to determine the correct rotation axis and angle. In our disentangled representation, rotation axis and angle are independent of the object



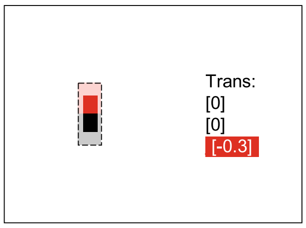
**(a)** Camera coord. xy-plane translation



**(b)** Disentangled coord. xy-plane translation



**(c)** Camera coord. z-axis translation



**(d)** Disentangled coord. z-axis translation

**Fig. 5** Translations using camera and our disentangled representations. In camera coordinates, translations in the image plane are represented by vectors in 3D space. As a result, the same translation in the 2D image corresponds to different translation vectors depending on whether an object is close or far from the camera. In our disentangled representation, the value of x and y is only related to the 2D vector in the image-plane. Additionally, as shown in (**c**), in the camera representation, a translation along the z-axis is not only difficult to infer from the image, but also causes a move relative to the center of the image. In our disentangled translation representation (**d**), only the change of scale needs to be estimated, making it independent of other translations and the metric size and distance of the object

To eliminate these problems, we propose to decouple the estimation of $\mathbf{R}_\Delta$ and $\mathbf{t}_\Delta$. First, we move the center of rotation from the origin of the camera to the center of the object in the camera frame, given by the current pose estimate. In this representation, a rotation does not change the translation of the object in the camera frame. The remaining question is how to choose the directions of the rotational axes of the coordinate frame. One way is to use the axes as specified in the 3D object model. However, as illustrated in column (c) of Fig. 4, such a representation would require the network to learn and memorize the coordinate frames of each object, which makes training more difficult and cannot be generalized to pose matching of unseen objects. Thus, we propose to use axes parallel to the axes of the camera frame when computing the relative rotation. By doing so, the network can be trained to estimate the relative rotation independently of the coordinate frame of the 3D object model, as illustrated in column (d) in Fig. 4.

In order to estimate the relative translation, let $\mathbf{t}_{\text{tgt}} = (x_{\text{tgt}}, y_{\text{tgt}}, z_{\text{tgt}})$ and $\mathbf{t}_{\text{src}} = (x_{\text{src}}, y_{\text{src}}, z_{\text{src}})$ be the target translation and the source translation. A straightforward way to represent translation is $\mathbf{t}_\Delta = (\Delta_x, \Delta_y, \Delta_z) = \mathbf{t}_{\text{tgt}} - \mathbf{t}_{\text{src}}$. However, it is not easy for the network to estimate the relative translation in the 3D metric space given only 2D images without depth information. The network has to recognize the size of the object, and map the translation in 2D space to 3D according to the object size. Such a representation is not only difficult for the network to learn, but also has problems when dealing with unknown objects or objects with similar appearance but different sizes. Instead of training the network to directly regress to the vector in the 3D space, we propose to regress to object changes in the 2D image space as shown in Fig. 5. Specifically, we train the network to regress to the relative translation $\mathbf{t}_\Delta = (v_x, v_y, v_z)$, where $v_x$ and $v_y$ denote the number of pixels the object should move along the image x-axis and y-axis and $v_z$ is the scale change of the object:

$$\begin{aligned} v_x &= f_x(x_{\text{tgt}}/z_{\text{tgt}} - x_{\text{src}}/z_{\text{src}}), \\ v_y &= f_y(y_{\text{tgt}}/z_{\text{tgt}} - y_{\text{src}}/z_{\text{src}}), \\ v_z &= \log(z_{\text{src}}/z_{\text{tgt}}), \end{aligned} \tag{3}$$

where $f_x$ and $f_y$ denote the focal lengths of the camera. The scale change $v_z$ is defined to be independent of the absolute object size or distance by using the ratio between the distances of the rendered and observed object. We use logarithm for $v_z$ to make sure that a value of zero corresponds to no change in scale or distance. Considering the fact that $f_x$ and $f_y$ are constant for a specific dataset, we simply fix it to 1 in training and testing the network.

Our representation of the relative transformation has several advantages. First, rotation does not influence the estimation of translation, so that the translation no longer needs to offset the movement caused by rotation around the camera center. Second, the intermediate variables $v_x$, $v_y$, $v_z$ represent simple translations and scale change in the image space. Third, this representation does not require any prior knowledge of the object. Using such a representation, the DeepIM network can operate independently of the actual size of the object, its internal model coordinate framework, and the camera intrinsics. It only has to learn to transform the rendered image such that it becomes more similar to the observed image (Fig. 5).

## 3.4 Matching Loss

A straightforward way to train the pose estimation network is to use separate loss functions for rotation and translation. For example, we can use the angular distance between two rotations to measure the rotation error and use the $\ell_2$ distance to measure the translation error. However, using two different loss functions for rotation and translation suffers from the difficulty of balancing the two losses. Kendall and

Cipolla (2017) proposed a geometric reprojection error as the loss function for pose regression that computes the average distance between the 2D projections of 3D points in the scene using the ground truth pose and the estimated pose. Considering the fact that we want to accurately predict the object pose in 3D, we introduce a modified version of the geometric reprojection loss in Kendall and Cipolla (2017), and we call it the Point Matching Loss. Given the ground truth pose $\mathbf{p} = [\mathbf{R}|\mathbf{t}]$ and the estimated pose $\hat{\mathbf{p}} = [\hat{\mathbf{R}}|\hat{\mathbf{t}}]$, the point matching loss is computed as:

$$L_{\text{pose}}(\mathbf{p}, \hat{\mathbf{p}}) = \frac{1}{n} \sum_{i=1}^{n} \|(\mathbf{R}\mathbf{x}_i + \mathbf{t}) - (\hat{\mathbf{R}}\mathbf{x}_i + \hat{\mathbf{t}})\|_1, \tag{4}$$

where $\mathbf{x}_i$ denotes a randomly selected 3D point on the object model and $n$ is the total number of points (we choose 3000 points in our experiments). The formulation of point matching loss is similar to the one used to compute average distance (ADD) metric in Eq. 5. The main difference is that other than using $\ell_2$ norm, point matching loss computes the average $\ell_1$ distance between 3D points transformed by the ground truth pose and the estimated pose in order to avoid the large graident caused by outliers and maintain the stability of loss during training. In this way, it measures how the transformed 3D models match against each other for pose estimation. Xiang et al. (2018) also uses a variant of the point matching loss for rotation regression.

### 3.5 Training and Testing

In training, we assume that we have 3D object models and images annotated with ground truth 6D object poses. By adding noises to the ground truth poses as the initial poses, we can generate the required observed and rendered inputs to the network along with the pose target output that is the pose difference between the ground truth pose and the noisy pose. Then we can train the network to predict the relative transformation between the initial pose and the target pose.

During testing, we find that the iterative pose refinement can significantly improve the accuracy. To see, let $\mathbf{p}^{(i)}$ be the pose estimate after the $i$-th iteration of the network. If the initial pose estimate $\mathbf{p}^{(0)}$ is relatively far from the correct pose, the rendered image $\mathbf{img}_{\text{rend}}(\mathbf{p}^{(0)})$ may have only little viewpoint overlap with the observed image $\mathbf{img}_{\text{obs}}$. In such cases, it is very difficult to accurately estimate the relative pose transformation $\mathbf{1p}^{(0)}$ directly. This task is even harder if the network has no priori knowledge about the object to be matched. In general, it is reasonable to assume that if the network improves the pose estimate $\mathbf{p}^{(i+1)}$ by updating $\mathbf{p}^{(i)}$ with $\mathbf{1p}^{(i)}$ in the $i$-th iteration, then the image rendered according to this new estimate, $\mathbf{img}_{\text{rend}}(\mathbf{p}^{(i+1)})$ is also more similar to the observed image $\mathbf{img}_{\text{obs}}$ than $\mathbf{img}_{\text{rend}}(\mathbf{p}^{(i)})$ was in the previous iteration, thereby providing input that can be matched more accurately.

However, we found that, if we train the network to regress the relative pose in a single step, the estimates of the trained network do not improve over multiple iterations in testing. To generate a more realistic data distribution for training similar to testing, we perform multiple iterations during training as well. Specifically, for each training image and pose, we apply the transformation predicted from the network to the pose and use the transformed pose estimate as another training example for the network in the next iteration. By repeating this process multiple times, the training data better represents the test distribution and the trained network also achieves significantly better results during iterative testing [such an approach has also proven useful for iterative hand pose matching (Oberweger et al. 2015) and image alignment (Lin and Lucey 2017)].

## 4 Experiments

We conduct extensive experiments on the LINEMOD dataset (Hinterstoisser et al. 2012b) and the Occlusion LINEMOD dataset (Brachmann et al. 2014) to evaluate our DeepIM framework for 6D object pose estimation. We test different properties of DeepIM and show that it surpasses other RGB-only methods by a large margin. We also show that our network can be applied to pose matching of unseen objects during training.

### 4.1 Training Implementation Details

**Training Parameters:** We use the pre-trained FlowNetSimple (Dosovitskiy et al. 2015) to initialize the weights in our network. Weights of the new layers are randomly initialized, except for the additional weights in the first conv layer that deals with the input masks and the fully-connected layer that predicts the translation, which are initialized with zeros. Other than predicting the pose transformation, the network also predicts the optical flow and the foreground mask. Including the two additional losses could slightly increase the pose estimation performance and make the training more stable. Specifically, we use the optical flow loss $L_{\text{flow}}$ as in FlowNet (Dosovitskiy et al. 2015) and the sigmoid cross-entropy loss as the mask loss $L_{\text{mask}}$. Two deconvolutional blocks in FlowNet are inherited to produce the feature map used for the mask and the optical flow prediction, whose spatial scale is 0.0625. Two $1 \times 1$ convolutional layers with output channel 1 (mask prediction) and 2 (flow prediction) are appended after this feature map. The predictions are then

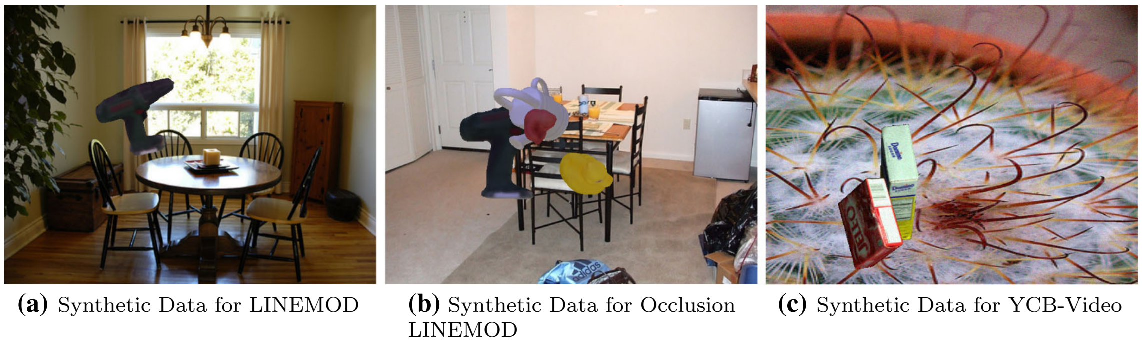

(a) Synthetic Data for LINEMOD
(b) Synthetic Data for Occlusion LINEMOD
(c) Synthetic Data for YCB-Video

**Fig. 6** Synthetic Data for the LINEMOD, Occlusion LINEMOD and YCB-Video separately. **a** Synthetic training data used when training on the LINEMOD dataset, only one object is presented in the image so there is no occlusion. **b** Synthetic training data used when training on the Occlusion LINEMOD dataset, multiple objects are presented in one image so one object may be occluded by other objects. **c** Synthetic training data used when training on the YCB-Video dataset. These images are rendered on the fly, so we only render two objects to maintain efficiency

bilinearly up-sampled to the original image size (480 × 640) to compute losses.

The overall loss is $L = \alpha L_{\text{pose}} + \beta L_{\text{flow}} + \gamma L_{\text{mask}}$, where we use $\alpha = 0.1$, $\beta = 0.25$, $\gamma = 0.03$ throughout the experiments (except some of our ablation studies). Each training batch contains 16 images. We train the network with 4 GPUs where each GPU processes 4 images. We generate 4 items for each image as described in Sect. 3.1: two images and two masks. The observed mask is randomly dilated with no more than 10 pixels to avoid over-fitting.

**The Distribution of Rendered Pose During Training:** The rendered image $\mathbf{img}_{\text{rend}}$ and mask $\mathbf{m}_{\text{rend}}$ are randomly generated during training without using prior knowledge of the initial poses in the test set. Specifically, given a ground truth pose $\hat{\mathbf{p}}$, we add noises to $\hat{\mathbf{p}}$ to generate the rendered poses. For rotation, we independently add a Gaussian noise $\mathcal{N}(0, 15^2)$ to each of the three Euler angles of the rotation. If the angular distance between the new pose and the ground truth pose is more than 45°, we discard the new pose and generate another one in order to make sure the initial pose for refinement is within 45° of the ground truth pose during training. For translation, considering the fact that RGB-based pose estimation methods usually have larger standard deviation on depth estimation, the following Gaussian noises are added to the three components of the translation: $\Delta x \sim \mathcal{N}(0, 0.01^2)$, $\Delta y \sim \mathcal{N}(0, 0.01^2)$, $\Delta z \sim \mathcal{N}(0, 0.05^2)$, where the standard deviations are 1 cm, 1 cm and 5 cm, respectively.

**Synthetic Training Data:** Real training images provided in existing datasets may be highly correlated or lack images in certain situations such as occlusions between objects. Therefore, generating synthetic training data is essential to enable the network to deal with different scenarios in testing. In generating synthetic training data for the LINEMOD dataset, considering the fact that the elevation variation is limited in this dataset, we calculate the elevation range of the objects in the provided training data. Then we rotate the object model with a randomly generated quaternion and repeat it until the elevation is within this range. The translation is randomly generated using the mean and the standard deviation computed from the training set. During training, the background of the synthetic image is replaced by a randomly chosen indoor image from the PASCAL VOC dataset as shown in Fig. 6.

For the Occlusion LINEMOD dataset, multiple objects are rendered into one image in order to introduce occlusions among objects. The number of objects ranges from 3 to 8 in these synthetic images. As in the LINEMOD dataset, the quaternion of each object is also randomly generated to ensure that the elevation range is within that of training data in the Occlusion LINEMOD dataset. The translations of the objects in the same image are drawn according to the distributions of the objects in the YCB-Video dataset (Xiang et al. 2018) by adding a small Gaussian noise.

For the YCB-Video dataset, synthetic images are generated on the fly. Other than the target object, we also render another object close to it to introduce partial occlusion.

The real training images may also lack variations in light conditions exhibited in the real world or in the testing set. Therefore, we add a random light condition to each synthetic image in both the LINEMOD dataset and the Occlusion LINEMOD dataset.

### 4.2 Testing Implementation Details

**Testing Parameters:** The mask prediction branch and the optical flow branch are removed during testing. Since there is no ground truth segmentation of the object in testing, we use the tightest bounding box of the rendered mask $\mathbf{m}_{\text{rend}}$

instead, so the network searches the neighborhood near the estimated pose to find the target object to match. Unless specified, we use the pose estimates from PoseCNN (Xiang et al. 2018) as the initial poses. Our DeepIM network runs at 12 fps per object using an NVIDIA 1080 Ti GPU with 2 iterations during testing.

**Pose Initialization During Inference:** Our framework takes an input image and an initial pose estimation of an object in the image as inputs, and then refine the initial pose iteratively. In our experiments, we have tested two pose initialization methods.

The first one is PoseCNN (Xiang et al. 2018), a convolutional neural network designed for 6D object pose estimation. PoseCNN performs three tasks for 6D pose estimation, i.e., semantic labeling to classify image pixels into object classes, localizing the center of the object on the image to estimate the 3D translation of the object, and 3D rotation regression. In our experiments, we use the 6D poses from PoseCNN as initial poses for pose refinement.

To demonstrate the robustness of our framework on pose initialization, we have implemented a simple 6D pose estimation method for pose initialization, where we extend the Faster R-CNN framework designed for 2D object detection (Ren et al. 2015) to 6D pose estimation. Specifically, we use the bounding box of the object from Faster R-CNN to estimate the 3D translation of the object. The center of the bounding box is treated as the center of the object. The distance of the object is estimated by maximizing the overlap of the projection of the 3D object model with the bounding box. To estimate the 3D rotation of the object, we add a rotation regression branch to Faster R-CNN as in PoseCNN. In this way, we can obtain a 6D pose estimation for each detected object from Faster R-CNN.

In our experiments on the LINEMOD dataset described in Sect. 4.4, we have shown that, although the initial poses from Faster R-CNN are much worse than the poses from PoseCNN, our framework is still able to refine these poses using the *same* weights. The performance gap between using the two different pose initialization methods is quite small, which demonstrates the ability of our framework in using different methods for pose initialization.

## 4.3 Evaluation Metrics

We use the following three evaluation metrics for 6D object pose estimation. (1) The 5°, 5 cm metric considers an estimated pose to be correct if its rotation error is within 5° and the translation error is below 5 cm. (2) The *6D Pose* metric (Hinterstoisser et al. 2012b) computes the average distance between the 3D model points transformed using the estimated pose and the ground truth pose. For symmetric objects, we use the closest point distance in computing the average distance. An estimated pose is correct if the average distance is within 10% of the 3D model diameter. (3) The *2D Projection* metric computes the average distance of the 3D model points projected onto the image using the estimated pose and the ground truth pose. An estimated pose is correct if the average distance is smaller than 5 pixels.
k°, k cm: Proposed in Shotton et al. (2013). The 5°, 5 cm metric considers an estimated pose to be correct if its rotation error is within 5° and the translation error is below 5 cm. We also provided the results with 2°, 2 cm and 10°, 10 cm in Table 6 to give a comprehensive view about the performance.

For symmetric objects such as eggbox and glue in the LINEMOD dataset, we compute the rotation error and the translation error against all possible ground truth poses with respect to symmetry and accept the result when it matches one of these ground truth poses.

**6D Pose:** Hinterstoisser et al. (2012b) use the average distance (ADD) metric to compute the averaged distance between points transformed using the estimated pose and the ground truth pose as in Eq. 5:

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

where $m$ is the number of points on the 3D object model, $\mathcal{M}$ is the set of all 3D points of this model, $\mathbf{p} = [\mathbf{R}|\mathbf{t}]$ is the ground truth pose and $\hat{\mathbf{p}} = [\hat{\mathbf{R}}|\hat{\mathbf{t}}]$ is the estimated pose. Here the number of points $m$ can be different from the number of points $n$ used in Eq. 4 as the point clouds used for training is a subset randomly sampled from the original point clouds to reduce the time to compute the loss during training. $\mathbf{Rx} + \mathbf{t}$ indicates transforming the point with the given SE(3) transformation (pose) $\mathbf{p}$. Following Brachmann et al. (2016), we compute the distance between all pairs of points from the model and regard the maximum distance as the diameter $d$ of this model. Then a pose estimation is considered to be correct if the computed average distance is within 10% of the model diameter. In addition to using $0.1d$ as the threshold, we also provided pose estimation accuracy using thresholds $0.02d$ and $0.05d$ in Table 6. We use $0.1d$ as the threshold of 6D Pose metric in the following paper if not specified.

For symmetric objects, we use the closest point distance in computing the average distance for 6D pose evaluation as in Hinterstoisser et al. (2012b):

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

In the YCB-Video Dataset, we use the metric ADD and ADD-S described in Xiang et al. (2018). After getting the

ADD and ADD-S distance described in Eqs. 5 and 6, we vary the threshold from 0 to 10 cm and accumulate the area under the accuracy curves.

**2D Projection:** focuses on the matching of pose estimation on 2D images. This metric is considered to be important for applications such as augmented reality. We compute the error using Eq. 7 and accept a pose estimation when the 2D projection error is smaller than a predefined threshold:

$$\textbf{Proj. 2D} = \frac{1}{m} \sum_{x \in \mathcal{M}} \|\mathbf{K}(\mathbf{R}\mathbf{x} + \mathbf{t}) - \mathbf{K}(\hat{\mathbf{R}}\mathbf{x} + \hat{\mathbf{t}})\|_2, \tag{7}$$

where $\mathbf{K}$ denotes the intrinsic parameter matrix of the camera and $\mathbf{K}(\mathbf{R}\mathbf{x}+\mathbf{t})$ indicates transforming a 3D point according to the SE(3) transformation and then projecting the transformed 3D point onto the image. In addition to using 5 pixels as the threshold, we also show our results with the thresholds 2 pixels and 10 pixels. We use 5 pixels as the threshold of Proj. 2D metric in the following paper if not specified.

For symmetric objects such as eggbox and glue in the LINEMOD dataset, we compute the 2D projection error against all possible ground truth poses and accept the result when it matches one of these ground truth poses.

## 4.4 Experiments on the LINEMOD Dataset

The LINEMOD dataset contains 15 objects. We train and test our method on 13 of them as other methods in the literature. We follow the procedure in Brachmann et al. (2016) to split the dataset into the training and test sets, with around 200 images for each object in the training set and 1000 images in the test set. Figure 8 shows a subset of objects used in LINEMOD dataset. These objects are textureless and thus difficult for pose estimation methods using only local features.

**Training Strategy:** For every image, we generate 10 random poses near the ground truth pose, resulting in 2000 training samples for each object in the training set. Furthermore, we generate 10,000 synthetic images for each object where the pose distribution is similar to the real training set. For each synthetic image, we generate 1 random pose near its ground truth pose. Thus, we have a total of 12,000 training samples for each object in training. The background of a synthetic image is replaced with a randomly chosen indoor image from PASCAL VOC (Everingham et al. 2010). We train the networks for 8 epochs with initial learning rate 0.0001. The learning rate is divided by 10 after the 4th and 6th epoch, respectively.

**Ablation Study on Iterative Training and Testing:** Table 2 shows the results that use different numbers of iterations during training and testing. The networks with $train\_iter = 1$ and $train\_iter = 2$ are trained with 32 and 16 epochs respectively to keep the total number of updates the same as $train\_iter = 4$. The table shows that without iterative training ($train\_iter = 1$), multiple iteration testing does not improve, potentially even making the results worse ($test\_iter = 4$). We believe that the reason is due to the fact that the network is not trained with enough rendered poses close to their ground truth poses. The table also shows that one more iteration during training and testing already improves the results by a large margin. The network trained with 2 iterations and tested with 2 iterations is slightly better than the one trained with 4 iterations and tested with 4 iterations. This may be because the LINEMOD dataset is not sufficiently difficult to generate further improvements by using 3 or 4 iterations. Since it is not straightforward to determine how many iterations to use in each dataset, we use 4 iterations during training and testing in all other experiments.

**Ablation Study on the Zoom in Strategy, Network Structures, Transformation Representations, and Loss Functions:** Table 3 summarizes the ablation studies on various aspects of DeepIM. The "zoom" column indicates whether the network uses full images as its input or zoomed in bounding boxes up-sampled to the original image size. Comparing rows 5 and 7 shows that the higher resolution achieved via zooming in provides very significant improvements.

"Regressor": We train the DeepIM network jointly over all objects, generating a pose transformation independent of the specific input object (labeled "shared" in "regressor" column). Alternatively, we could train a different 6D pose regressor for each individual object by using a separate fully

**Table 2** Ablation study of the number of iterations during training and testing

| Train iter | Init | 1 | | | 2 | | | 4 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Test iter | | 1 | 2 | 4 | 1 | 2 | 4 | 1 | 2 | 4 |
| 5 cm 5° | 19.4 | 57.4 | 58.8 | 54.6 | 76.3 | 86.2 | 86.7 | 70.2 | 83.7 | 85.2 |
| 6D Pose | 62.7 | 77.9 | 79.0 | 76.1 | 83.1 | 88.7 | 89.1 | 80.9 | 87.6 | 88.6 |
| Proj. 2D | 70.2 | 92.4 | 92.6 | 89.7 | 96.1 | 97.8 | 97.6 | 94.6 | 97.4 | 97.5 |

**Table 3** Ablation study on different design choices of the DeepIM network on the LINEMOD dataset

| Row | Methods | | | | | 5 cm 5° | 6D Pose | Proj. 2D |
|---|---|---|---|---|---|---|---|---|
| | Zoom | Regressor | Network | Coordinate | Loss | | | |
| 1 | ✓ | – | Sep. | Disentangled | PM | 83.3 | 87.6 | 96.2 |
| 2 | ✓ | Sep. | Shared | Model | PM | 79.2 | 87.5 | 95.4 |
| 3 | ✓ | Sep. | Shared | Disentangled | PM | 86.6 | 89.5 | 96.7 |
| 4 | | Shared | Shared | Camera | PM | 16.6 | 44.3 | 62.5 |
| 5 | | Shared | Shared | Disentangled | PM | 38.3 | 65.2 | 80.8 |
| 6 | ✓ | Shared | Shared | Disentangled | Dist | 86.5 | 79.2 | 96.2 |
| 7 | ✓ | Shared | Shared | Disentangled | PM | 85.2 | 88.6 | 97.5 |

**Table 4** Ablation study on two different methods for generating initial poses on the LINEMOD dataset

| Method | PoseCNN | PoseCNN + Ours | Faster R-CNN | Faster R-CNN + Ours |
|---|---|---|---|---|
| 5 cm 5° | 19.4 | 85.2 | 11.9 | 83.4 |
| 6D Pose | 62.7 | 88.6 | 33.1 | 86.9 |
| Proj. 2D | 70.2 | 97.5 | 20.9 | 95.7 |

connected layer for each object after the final FC256 layer shown in Fig. 3. This setting is labeled as "sep." in Table 3. Comparing rows 3 and 7 shows that both approaches provide nearly indistinguishable results. But the shared network provides some efficiency gains.

"Network": Similarly, instead of training a single network over all objects, we could train separate networks, one for each object as in Rad and Lepetit (2017). Comparing row 1 to 7 shows that a single, shared network provides better results than individual ones, which indicates that training on multiple objects can help the network learn a more general representation for matching. We also present an ablation study of mask prediction and flow prediction in Table 1. It shows that when trained with these two auxiliary branches, the network could achieve the highest performance.

"Coordinate": This column investigates the impact of our choice of coordinate frame to reason about object transformations, as described in Fig. 4. The row labeled "camera" provides results when choosing the camera frame of reference as the representation for the object pose, rows labeled "model" move the center of rotation to the object model and choose the object model coordinate frame to reason about rotations, and the "disentangled" rows provide our disentangled approach of moving the center into the object model while keeping the camera coordinate frame for rotations. Comparing rows 2 and 3 shows that reasoning in the camera rotation frame provides slight improvements. Furthermore, it should be noted that only our "disentangled" approach is able to operate on unseen objects. Comparing rows 4 and 5 shows the large improvements our representation achieves over the common approach of reasoning fully in the camera frame of reference.

"Loss": The traditional loss for pose estimation is specified by the distance ("Dist") between the estimated and ground truth 6D pose coordinates, i.e., angular distance for rotation and euclidean distance for translation. Comparing rows 6 and 7 indicates that our point matching loss ("PM") provides significantly better results especially on the 6D pose metric, which is the most important measure for reasoning in 3D space.

**Application to Different Initial Pose Estimation Networks:** Table 4 provides results when we initialize DeepIM with two different pose estimation networks. The first one is PoseCNN (Xiang et al. 2018), and the second one is a simple 6D pose estimation method based on Faster R-CNN (Ren et al. 2015). Specifically, we use the bounding box of the object from Faster R-CNN to estimate the 3D translation of the object. The center of the bounding box is treated as the center of the object. The distance of the object is estimated by maximizing the overlap of the projection of the 3D object model with the bounding box. To estimate the 3D rotation of the object, we add a rotation regression branch to Faster R-CNN as in PoseCNN. As we can see in Table 4, our network achieves very similar pose estimation accuracy even when initialized with the estimates from the extension of Faster R-CNN, which are not as accurate as those provided by PoseCNN (Xiang et al. 2018).

**Comparison with the State-of-the-Art 6D Pose Estimation Methods:** Table 5 shows the comparison with the best color-only techniques on the LINEMOD dataset. DeepIM achieves very significant improvements over all prior methods, even those that also deploy refinement steps [BB8 (Rad and Lepetit 2017) and SSD-6D (Kehl et al. 2017)].

**Table 5** Comparison with state-of-the-art methods on the LINEMOD dataset

| Methods | Brachmann et al. (2016) | BB8 w/ref. (Rad and Lepetit 2017) | SSD-6D w/ref. (Kehl et al. 2017) | Tekin et al. (2017) | PoseCNN (Xiang et al. 2018) | PoseCNN (Xiang et al. 2018) + Ours |
|---|---|---|---|---|---|---|
| 5 cm 5° | 40.6 | 69.0 | – | – | 19.4 | **85.2** |
| 6D Pose | 50.2 | 62.7 | 79 | 55.95 | 62.7 | **88.6** |
| Proj. 2D | 73.7 | 89.3 | – | 90.37 | 70.2 | **97.5** |

The bold numbers indicate highest number in this comparison

**Table 6** Results of using more detailed thresholds on the LINEMOD dataset

| Metric threshold | (n°, n cm) | | | 6D Pose | | | Projection 2D | | |
|---|---|---|---|---|---|---|---|---|---|
| | (2, 2) | (5, 5) | (10,10) | 0.02d | 0.05d | 0.10d | 2 px. | 5 px. | 10 px. |
| Ape | 37.7 | 90.4 | 98.0 | 14.3 | 48.6 | 77.0 | 92.2 | 98.4 | 99.6 |
| Benchvise | 37.6 | 88.7 | 98.2 | 37.5 | 80.5 | 97.5 | 67.7 | 97.0 | 99.6 |
| Camera | 56.1 | 95.8 | 99.2 | 30.9 | 74.0 | 93.5 | 86.3 | 98.9 | 99.7 |
| Can | 58.0 | 92.8 | 99.0 | 41.4 | 84.3 | 96.5 | 98.6 | 99.7 | 99.8 |
| Cat | 33.5 | 87.6 | 97.8 | 17.6 | 50.4 | 82.1 | 88.4 | 98.7 | 100.0 |
| Driller | 49.4 | 92.9 | 99.1 | 35.7 | 79.2 | 95.0 | 64.2 | 96.1 | 99.4 |
| Duck | 30.8 | 85.2 | 98.5 | 10.5 | 48.3 | 77.7 | 88.1 | 98.5 | 99.8 |
| Eggbox | 32.1 | 63.9 | 94.5 | 34.7 | 77.8 | 97.1 | 53.4 | 96.2 | 99.6 |
| Glue | 32.8 | 83.0 | 98.0 | 57.3 | 95.4 | 99.4 | 81.5 | 98.9 | 99.7 |
| Holepuncher | 8.7 | 54.5 | 93.8 | 5.3 | 27.3 | 52.8 | 59.1 | 96.3 | 99.5 |
| Iron | 47.5 | 92.7 | 99.3 | 47.9 | 86.3 | 98.3 | 67.4 | 97.2 | 99.9 |
| Lamp | 47.5 | 90.9 | 98.4 | 45.3 | 86.8 | 97.5 | 60.0 | 94.2 | 99.0 |
| Phone | 34.8 | 89.6 | 98.6 | 22.7 | 60.5 | 87.7 | 75.9 | 97.7 | 99.8 |
| Mean | 39.0 | 85.2 | 97.9 | 30.9 | 69.2 | 88.6 | 75.6 | 97.5 | 99.7 |

**Detailed Results on the LINEMOD Dataset:** Table 6 shows our detailed results on all the 13 objects in the LINEMOD dataset. The network is trained and tested with 4 iterations and 8 epochs. Initial poses are estimated by PoseCNN (Xiang et al. 2018).

## 4.5 Experiments on the Occlusion LINEMOD Dataset

The Occlusion LINEMOD dataset proposed in Brachmann et al. (2014) shares the same images used in the LINEMOD dataset (Hinterstoisser et al. 2012b), but annotated 8 objects in one video that are heavily blocked by other objects.

**Training:** For every real image, we generate 10 random poses as described in Sect. 4.4. Considering the fact that most of the training data lacks occlusions, we generated about 20,000 synthetic images with multiple objects in each image. By doing so, every object has around 12,000 images which are partially occluded, and a total of 22,000 images for each object in training. We perform the same background replacement and training procedure as in the LINEMOD dataset.

**Comparison with the State-of-the-Art Methods:** The comparison between our method and other RGB-only methods is shown in Fig. 7. We only show the plots with accuracies on the 2D Projection metric because these are the only results reported in Rad and Lepetit (2017) and Tekin et al. (2017) (results for eggbox and glue use a symmetric version of this accuracy). It can be seen that our method greatly improves the pose accuracy generated by PoseCNN and surpasses all other RGB-only methods by a large margin. It should be noted that BB8 (Rad and Lepetit 2017) achieves the reported results only when using ground truth bounding boxes during testing. Our method is even competitive with the results that use depth information and ICP to refine the estimates of PoseCNN. Figure 8 shows some pose refinement results from our method on the Occlusion LINEMOD dataset.

**Detailed Results on the Occlusion LINEMOD Dataset:** Table 7 shows our results on the Occlusion LINEMOD dataset. We can see that DeepIM can significantly improve the initial poses from PoseCNN. Notice that the diameter here is computed using the extents of the 3D model follow-

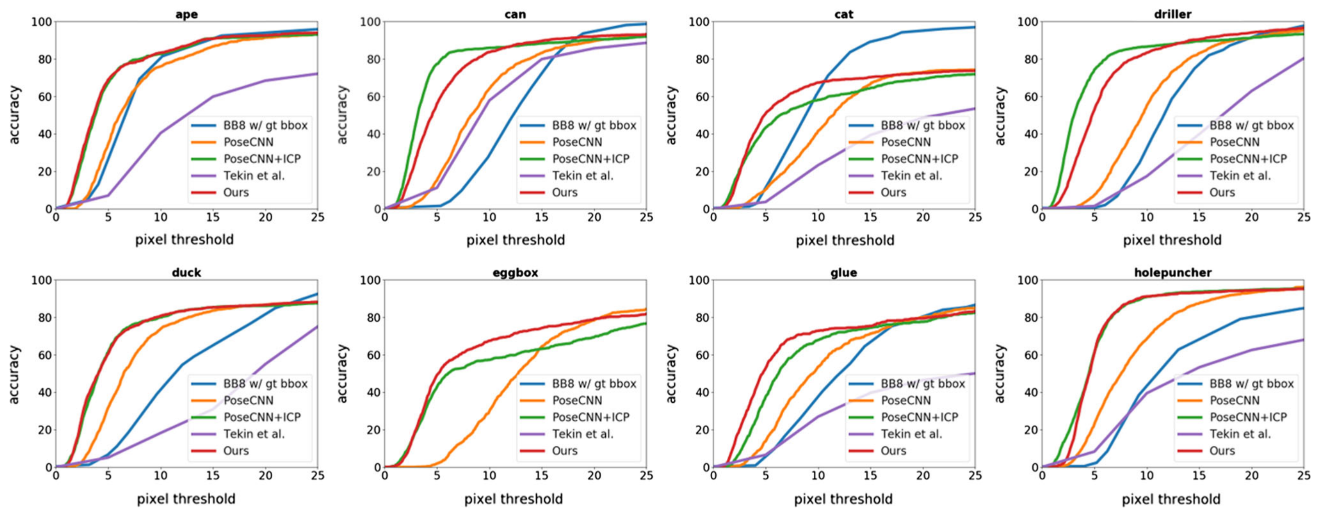


**Fig. 7** Comparison with state-of-the-art methods on the Occlusion LINEMOD dataset (Brachmann et al. 2014). Accuracies are measured via the Projection 2D metric

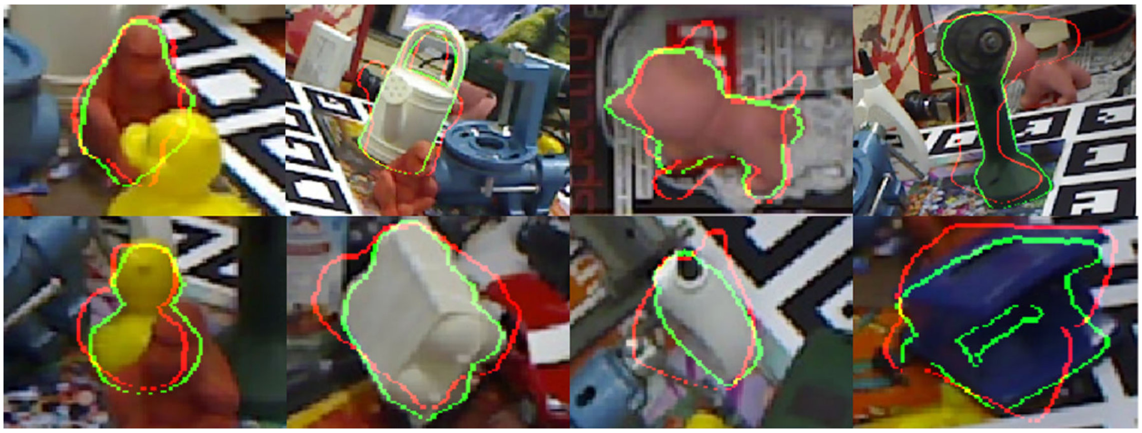


**Fig. 8** Examples of refined poses on the Occlusion LILNEMOD dataset using the results from PoseCNN (Xiang et al. 2018) as initial poses. The red and green lines represent the silhouettes of the initial estimates and our refined poses, respectively (Color figure online)

ing the setting of Xiang et al. (2018) and other RGB-D based methods. Some qualitative results are shown in Fig. 9.

### 4.6 Experiments on the YCB-Video Dataset

The YCB-Video Dataset, which is proposed in Xiang et al. (2018), annotates 21 YCB objects (Calli et al. 2015) in 92 video sequences (133,827 frames). It is a challenging dataset as the objects have varied sizes (diameter from 10 to 40 cm), different types of symmetries, and a large variety of occlusions and lighting conditions. We split the dataset as Xiang et al. (2018), with 80 video sequences for training and 2949 keyframes in the remaining 12 videos for testing.

**Training Strategy:** As images in one video are similar to those in nearby frames, we use 1 image out of every 10 images in the training set for training. Training batches consist of captured real images from the dataset (1/8) and synthetic images which are partially occluded and generated on the fly (7/8). The network is trained with 8 epochs and we decrease the learning rate after 4 and 6 epochs. We found that with large training sets and enough epochs it was not necessary to include the flow prediction and the masks in the input, so we removed those branches and the corresponding loss from this experiment. Again, we train all the categories with one network and shared regressor.

**Evaluation Metric:** We follow the PoseCNN (Xiang et al. 2018) paper when evaluating the results which uses accuracy under curve of ADD (Eq. 5) and ADD-S (Eq. 6 for each object. We also report the results of ADD(-S) and AUC ADD(-S) metric which is similar to the one we used in LINEMOD (Brachmann et al. 2014). More specifically, we use ADD when the object is not symmetric and use ADD-S

**Table 7** Results on the Occlusion LINEMOD dataset

| Metric | (5°, 5 cm) | | 6D Pose | | Projection 2D | |
|---|---|---|---|---|---|---|
| Method | Init. | Refined | Init. | Refined | Init. | Refined |
| Ape | 2.3 | **51.8** | 9.9 | **59.2** | 34.6 | **69.0** |
| Can | 4.1 | **35.8** | 45.5 | **63.5** | 15.1 | **56.1** |
| Cat | 0.3 | **12.8** | 0.8 | **26.2** | 10.4 | **50.9** |
| Driller | 2.5 | **45.2** | 41.6 | **55.6** | 7.4 | **52.9** |
| Duck | 1.8 | **22.5** | 19.5 | **52.4** | 31.8 | **60.5** |
| Eggbox | 0.0 | **17.8** | 24.5 | **63.0** | 1.9 | **49.2** |
| Glue | 0.9 | **42.7** | 46.2 | **71.7** | 13.8 | **52.9** |
| Hole. | 1.7 | **18.8** | 27.0 | **52.5** | 23.1 | **61.2** |
| Mean | 1.7 | **30.9** | 26.9 | **55.5** | 17.2 | **56.6** |

The bold numbers indicate highest number in this comparison
The network is trained and tested with 4 iterations

when the object is symmetric. Then we compute the averaged accuracy as the final result.

**Symmetric Objects:** As described in Sect. 4.1, we only keep rendered poses that have an angular distance less than 45° from ground truth poses during training, which means we don't need to take special care of objects which have a symmetry angle of more than 90°. However, object 024_bowl in the YCB-Video dataset is rotational symmetric. To deal with this kind of symmetry, rather than using the ground truth pose $\hat{\mathbf{p}}$ provided by the dataset to compute the loss, we choose the distance to the closest pose $\mathbf{p}^*$ among all poses that look the same as the ground truth pose:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}\in\mathcal{Q}} \Theta(\mathbf{p}, \mathbf{p}_{\text{src}}) \tag{8}$$

Here $\mathcal{Q}$ denotes the set of poses whose corresponding rendered images are the same as the one rendered using the ground truth pose. We assume that the rotation axis goes through the origin of the model frame so that no translation needs to be considered. In the experiment, we calibrate the rotation axis manually and use bisection search to locate the *closest ground truth pose*. Table 8 compares networks trained with and without this strategy, showing that this training loss is useful.

**Comparison with State-of-the-Art Methods:** Table 10 compares our results with two state-of-the-art methods:

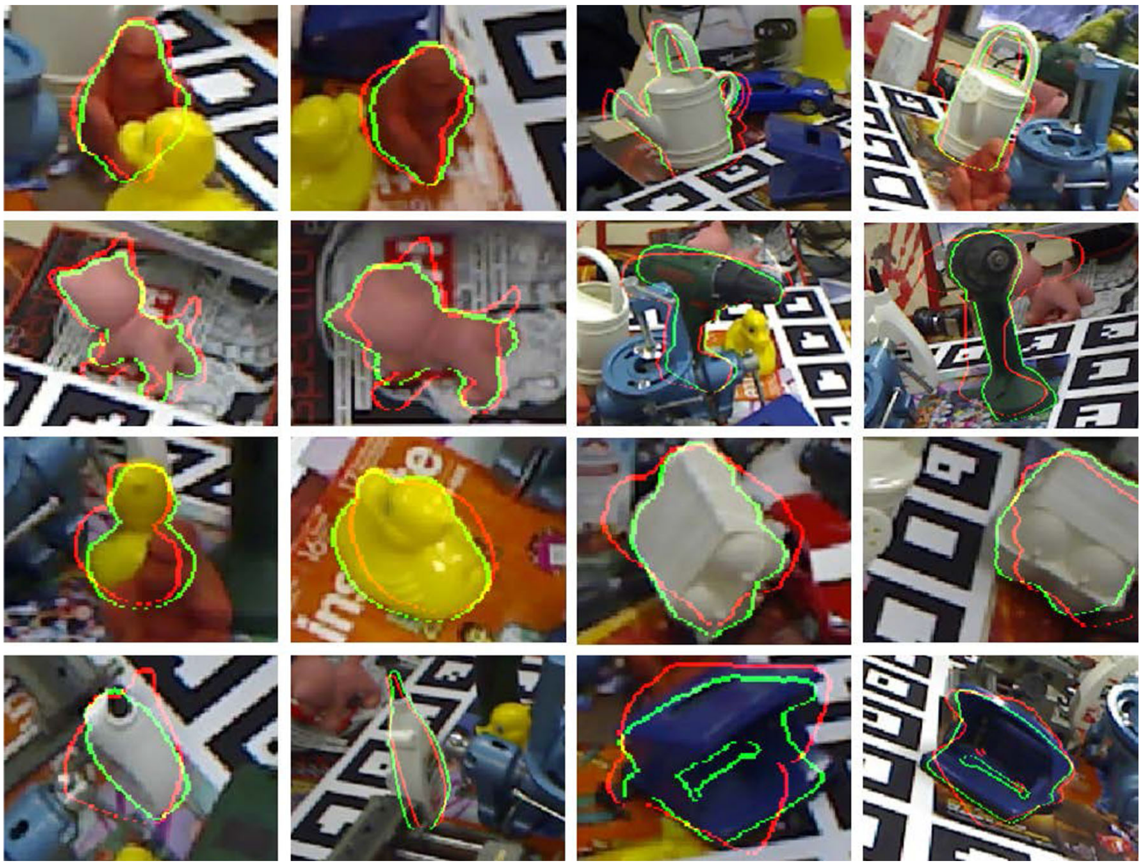


**Fig. 9** Some pose refinement results on the Occlusion LINEMOD dataset. The red and green lines represent the edges of 3D model projected from the initial poses and our refined poses respectively (Color figure online)

**Table 8** Ablation study about using *closest ground truth pose* to handle rotational symmetric objects

| 024_bowl | init | Common | Closest |
|---|---|---|---|
| ADD | 54.2 | 55.6 | 68.4 |
| ADD-S | 76.0 | 70.6 | 80.9 |

These three columns show the evaluation results of initial poses, poses refined by a DeepIM network that treats 024_bowl as a regular object, and poses refined by a network trained with *closest ground truth pose*. Initial poses are generated as rendered pose during training described in Sect. 4.1

PoseCNN (Xiang et al. 2018) and DenseFusion (Wang et al. 2019). As can be seen, DeepIM greatly refines the initial pose provided by PoseCNN and is on par with those refined with ICP on many objects despite not using any depth or point cloud data. Notice that DeepIM produces low numbers on symmetric objects, like 024_bowl, under ADD metric. This is because the ADD metric cannot well represent the performance on symmetric objects as such objects have multiple correct poses but only one of these poses are labeled as the ground truth in the dataset. Table 9 shows the result compared with PoseCNN (Xiang et al. 2018) and PoseRBPF (Deng et al. 2019) using the ADD(-S) metrci which can avoid such problems. Figure 10 visualizes some pose refinement results from our method on the YCB-Video dataset.

**Tracking in the YCB-Video Dataset:** Considering the similarity between pose refinement and object tracking, it is natural to use DeepIM  to track objects in videos. Therefore, we conducted an experiment testing DeepIM's ability

**Table 9** Overall results on YCB video results compared with PoseCNN (Xiang et al. 2018) and PoseRBPF (Deng et al. 2019)

| Methods | RGB | | | | RGB-D | | |
|---|---|---|---|---|---|---|---|
| | PoseCNN | PoseRBPF++ | PoseCNN + DeepIM | DeepIM+ Tracking | PoseCNN + ICP | PoseRBPF | PoseCNN + DeepIM |
| ADD(-S) $< 2$ cm | 27.55 | – | 71.5 | **79.0** | 78.9 | – | **90.3** |
| AUC of ADD(-S) | 61.31 | 64.4 | 81.9 | **85.9** | 86.6 | 88.5 | **90.4** |

The bold numbers indicate highest number in this comparison
The ADD(-S) metric and AUC ADD(-S) metric is introduced in Sect. 4.6

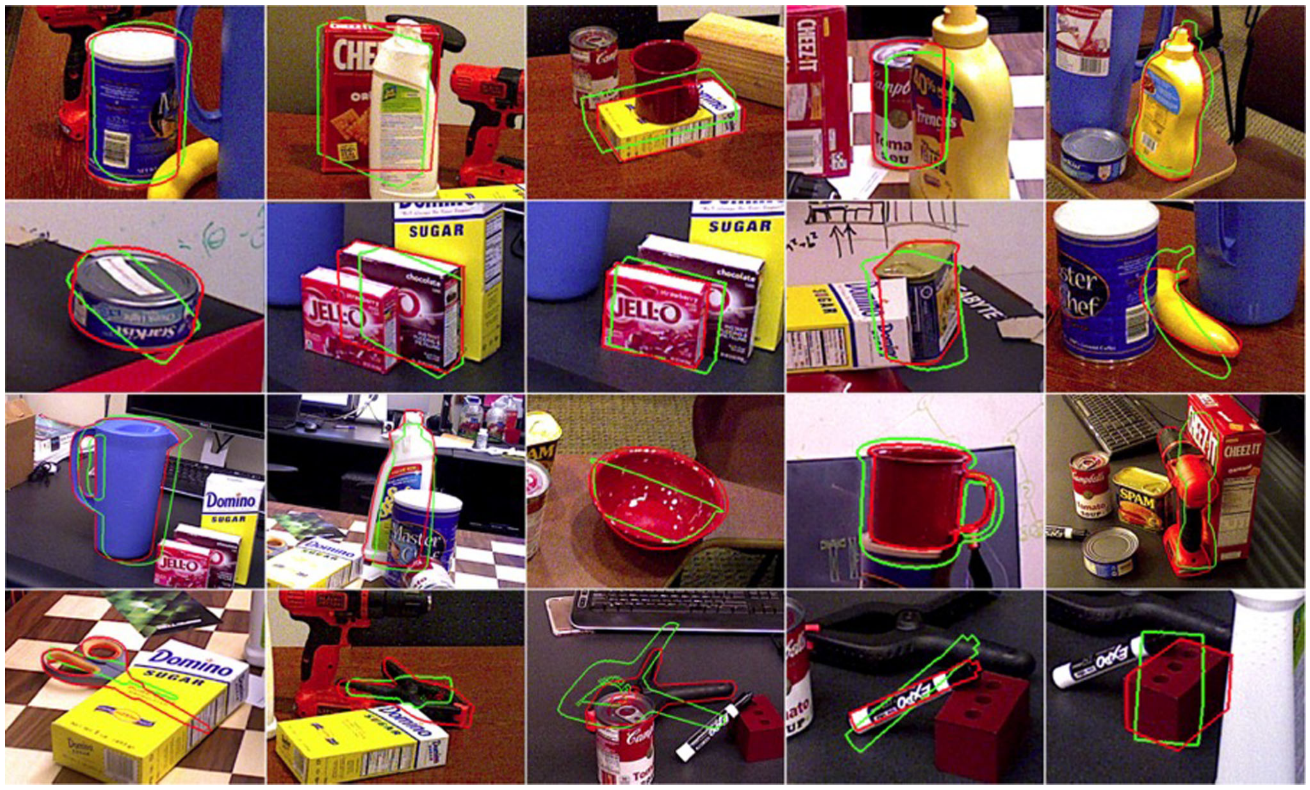


**Fig. 10** Examples of refined poses on the YCB-Video dataset which use results from PoseCNN (Xiang et al. 2018) as initial poses. The green and red lines represent the silhouettes of the initial estimates and our refined poses, respectively (Color figure online)

**Table 10** Detailed results on the YCB-video dataset compared with PoseCNN (Xiang et al. 2018) and DenseFusion (Wang et al. 2019)

| Methods | RGB | | | | | | RGB-D | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | PoseCNN | | PoseCNN + DeepIM | | DeepIMTracking | | PoseCNN + ICP | | DenseFusion | PoseCNN + DeepIM | |
| Evaluation metric | ADD | ADD-S | ADD | ADD-S | ADD | ADD-S | ADD | ADD-S | ADD-S | ADD | ADD-S |
| 002_master_chef_can | 50.2 | 83.9 | 71.2 | 93.1 | **89.0** | **93.8** | 68.1 | 95.8 | **96.4** | **78.0** | 96.3 |
| 003_cracker_box | 53.1 | 76.9 | 83.6 | 91.0 | **88.5** | **93.0** | 83.4 | 92.7 | **95.5** | **91.4** | 95.3 |
| 004_sugar_box | 68.4 | 84.3 | 94.1 | 96.2 | **94.3** | **96.3** | 97.2 | **98.2** | 97.5 | 97.6 | **98.2** |
| 005_tomato_soup_can | 66.2 | 81.0 | 86.1 | 92.4 | **89.1** | **93.2** | 81.8 | 94.5 | 94.6 | **90.3** | **94.8** |
| 006_mustard_bottle | 81.0 | 90.4 | 91.5 | **95.1** | **92.0** | **95.1** | **98.0** | 98.6 | 97.2 | 97.1 | **98.0** |
| 007_tuna_fish_can | 70.7 | 88.1 | 87.7 | 96.1 | **92.0** | **96.4** | 83.9 | 97.1 | 96.6 | **92.2** | **98.0** |
| 008_pudding_box | 62.7 | 79.1 | **82.7** | **90.7** | 80.1 | 88.3 | **96.6** | **97.9** | 96.5 | 83.5 | 90.6 |
| 009_gelatin_box | 75.2 | 87.2 | 91.9 | 94.3 | **92.0** | **94.4** | **98.1** | **98.8** | 98.1 | 98.0 | 98.5 |
| 010_potted_meat_can | 59.5 | 78.5 | 76.2 | 86.4 | **78.0** | **88.9** | **83.5** | **92.7** | 91.3 | 82.2 | 90.3 |
| 011_banana | 72.3 | 86.0 | **81.2** | **91.3** | 81.0 | 90.5 | 91.9 | 97.1 | 96.6 | **94.9** | **97.6** |
| 019_pitcher_base | 53.3 | 77.0 | 90.1 | 94.6 | **90.4** | **94.7** | 96.9 | 97.8 | 97.1 | **97.4** | **97.9** |
| 021_bleach_cleanser | 50.3 | 71.6 | 81.2 | 90.3 | **81.7** | **90.5** | **92.5** | **96.9** | 95.8 | 91.6 | **96.9** |
| **024_bowl** | 30.0 | 70.0 | 8.6 | 81.4 | **38.8** | **90.6** | **47.6** | 80.8 | **88.2** | 8.1 | 87.0 |
| 025_mug | 58.5 | 78.2 | 81.4 | 91.3 | **83.2** | **92.0** | 81.1 | 95.0 | 97.1 | **94.2** | **97.6** |
| 035_power_drill | 55.3 | 72.7 | **85.5** | **92.3** | 85.4 | **92.3** | **97.7** | **98.2** | 96.0 | 97.2 | 97.9 |
| **036_wood_block** | 26.6 | 64.3 | **60.0** | **81.9** | 44.3 | 75.4 | 70.9 | 87.6 | 89.7 | **81.1** | **91.5** |
| 037_scissors | 35.8 | 56.9 | 60.9 | 75.4 | **70.3** | **84.5** | 78.4 | 91.7 | 95.2 | **92.7** | **96.0** |
| 040_large_marker | 58.3 | 71.7 | 75.6 | 86.2 | **80.4** | **91.2** | 85.3 | 97.2 | 97.5 | **88.9** | **98.2** |
| **051_large_clamp** | 24.6 | 50.2 | 48.4 | 74.3 | **73.9** | **84.1** | 52.1 | 75.2 | 72.9 | **54.2** | **77.9** |
| **052_extra_large_clamp** | 16.1 | 44.1 | 31.0 | 73.3 | **49.3** | **90.3** | 26.5 | 64.4 | 69.8 | **36.5** | **77.8** |
| **061_foam_brick** | 72.9 | 88.2 | 35.9 | 81.9 | **91.6** | **95.5** | **90.5** | 97.4 | 92.5 | 48.2 | **97.6** |
| MEAN | 53.4 | 74.6 | 71.7 | 88.1 | **79.3** | **91.0** | 80.6 | 92.4 | 93.0 | **80.7** | **94.0** |

The bold numbers indicate symmetric objects

The network is trained and tested with 4 iterations. The ADD and ADD-S is short for AUC of ADD and AUC of ADD-S

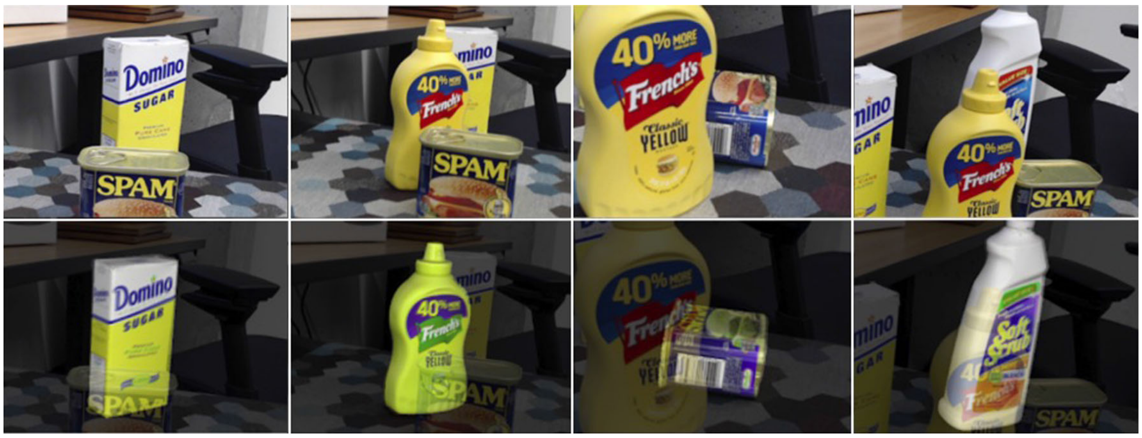


**Fig. 11** Examples on tracking in the real world, using the same network as in Table 10 and no prior knowledge about focal length. The first row shows the images captured with a webcam and the second row renders the object onto the image based on the estimated pose

to track objects in the YCB-Video dataset. Provided with the ground truth pose of an object in the first frame of each video, DeepIM can perform tracking by using the refined pose estimate from the previous frame as the initial pose of the next frame. Rather than doing inference only on key frames, we applied DeepIM to all images in the test video so that the object poses were close between successive frames.

In order to determine when DeepIM loses track of an object due to heavy occlusion, we follow a simple strategy: we count the tracking as "lost" if the last iteration of the last 10 frames has an average rotation greater than 10° or an average translation greater than 1 cm. Once the tracking is marked as lost, the network will be re-initialzed with PoseCNN's prediction. This strategy is designed with the intuition that successful tracking should have a small offset at the last iteration. Re-initialization happens every 340 frames on average. Tables 9 and 10 shows our numerical results. Notice that the results of tracking are better than PoseCNN + DeepIM in most cases and are comparable to the results refined with ICP which uses depth information. Also note that the performance on object 036_wood_block is bad because the model of the wooden block is different from the object used in the actual dataset video, which makes it nearly impossible to match the model with the image.

**Tracking YCB Objects in Real Scenes:** To demonstrate our framework's generalization, we use our network to track objects in real scenes. This means we don't have any prior knowledge about the lighting conditions, background, or camera parameters. Similar to tracking on the YCB-Video dataset, we use DeepIM to refine poses predicted from the previous frame. Thanks to the disentangled representation, we did not have to calibrate the camera to get its intrinsic matrix. Figure 11 shows some tracking results using our method in the real world environment in real time.

**Using Depth Information:** Other than using RGB images to do pose refinement, DeepIMcan be easily extended to utilize depth information to improve its performance. Here we append the depth images of the observed image and the rendered image with the two zero-initialized additional channels in the first convolution (one for the rendered depth and the other for the observed depth). To provide the network with information of the center of the object, we normalize the depth images by subtract them from the depth of the object's center. The results are shown in Table 10.

**Failure Cases:** In Fig. 12 we show 10 instances that the network fails to refine to a correct pose. They can be grouped into five categories: (1) discrepancy between object models and images. This can be caused by bad light conditions or an inaccurate object model; (2) few patterns to match. This usually happens when only certain featureless side-views are visible or the object is heavily occluded; (3) objects' shapes are unusual and difficult to learn; (4) the initial pose is too far away from the correct pose; (5) objects with tiny key components.

## 4.7 Application to Unseen Objects and Unseen Categories

As stated in Sect. 3.3, we designed the disentangled pose representation such that it is independent of the coordinate frame and the size of a specific 3D object model. In other words, the transformation predicted from the network does not need to have prior knowledge about the model itself. Therefore, the pose transformations correspond to operations in the image space. This opens the question whether DeepIM can refine the poses of objects that are not included in the training set. From the experiment results we found that our network can perform accurate refinement on these unseen models. See

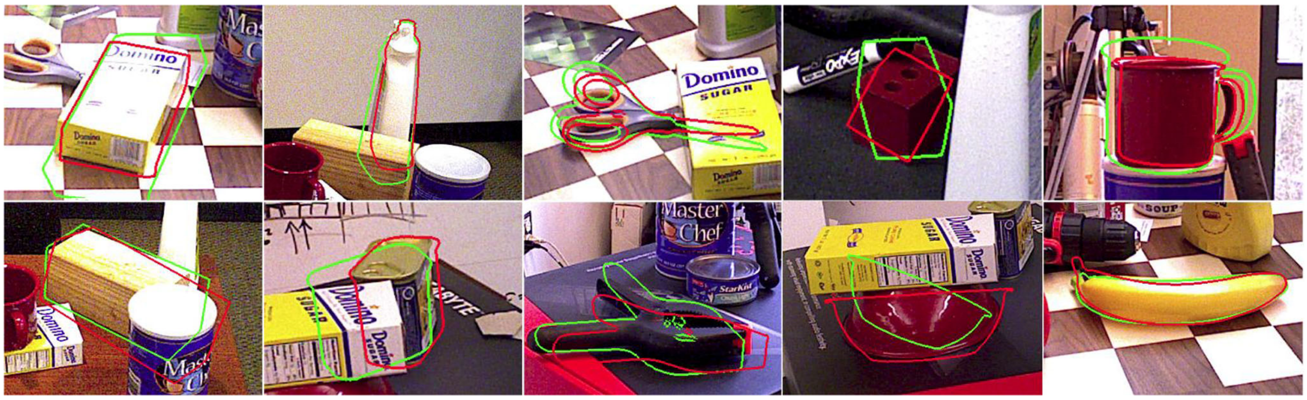


**Fig. 12** Failure cases in YCB-Video dataset. These images illustrate five different reasons we concluded that leads to fail cases

**Fig. 13** Results on pose refinement of 3D models from the ModelNet dataset. These instances were not seen in training. The red and green lines represent the edges of the initial estimates and our refined poses (Color figure online)

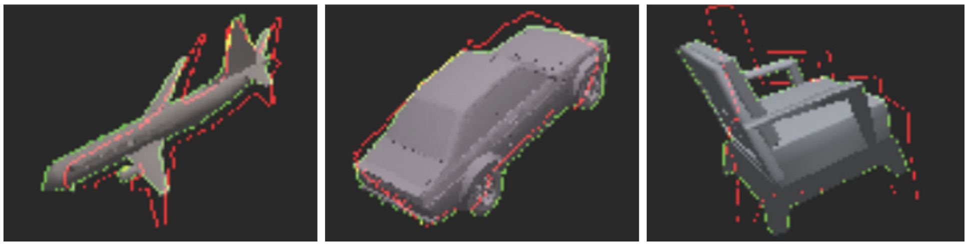


Fig. 13 for example results. We also tested our framework on refining the poses of unseen object categories, where the training categories and the test categories are completely different.

**Test on Unseen Objects:** In this experiment, we explore the ability of the network in refining poses of objects that has never been seen in training. ModelNet (Wu et al. 2015) contains a large number of 3D models in different object categories. Here, we tested our network on three of them: *airplane, car and chair*. For each of these categories, we train a network on no more than 200 3D models and test its performance on 70 unseen 3D models from the same category. Similar to the way that we generate synthetic data as described in Sec 4.1, we generate 50 poses for each model as the target poses and train the network for 4 epochs. We use uniform gray texture for each model and add a light source which has a fixed relative position to the object to reflect the norms of the object. The initial pose used in training and testing is generated in the same way as we did in previous experiments as described in Sect. 4.1. The results are show in Table 11.

**Table 11** Results on unseen objects

| Category | Airplane | | Car | | Chair | |
|---|---|---|---|---|---|---|
| Method | Init. | Refined | Init. | Refined | Init. | Refined |
| 5 cm 5° | 0.8 | **68.9** | 1.0 | **81.5** | 1.0 | **87.6** |
| 6D Pose | 25.7 | **94.7** | 10.8 | **90.7** | 14.6 | **97.4** |
| Proj. 2D | 0.4 | **87.3** | 0.2 | **83.9** | 1.5 | **88.6** |

The bold numbers indicate highest number in this comparison
These models are not included in the training set

**Table 12** Results on unseen categories

| Metric | (5°, 5 cm) | | 6D Pose | | Projection 2D | |
|---|---|---|---|---|---|---|
| Method | Init. | Refined | Init. | Refined | Init. | Refined |
| Bathtub | 0.9 | **71.6** | 11.9 | **88.6** | 0.2 | **73.4** |
| Bookshelf | 1.2 | **39.2** | 9.2 | **76.4** | 0.1 | **51.3** |
| Guitar | 1.2 | **50.4** | 9.6 | **69.6** | 0.2 | **77.1** |
| Range hood | 1.0 | **69.8** | 11.2 | **89.6** | 0.0 | **70.6** |
| Sofa | 1.2 | **82.7** | 9.0 | **89.5** | 0.1 | **94.2** |
| Wardrobe | 1.4 | **62.7** | 12.5 | **79.4** | 0.2 | **70.0** |
| TV stand | 1.2 | **73.6** | 8.8 | **92.1** | 0.2 | **76.6** |

The bold numbers indicate highest number in this comparison
These categories has never been seen by the network during training

**Test on Unseen Categories:** We also tested our framework on refining the poses of unseen object categories, where the training categories and the test categories are completely different. We train the network on 8 categories from ModelNet (Wu et al. 2015): *airplane, bed, bench, car, chair, piano, sink, toilet* with 30 models in each category and 50 image pairs for each model. The network was trained with 4 iterations and 4 epochs. Then we tested the network on 7 other categories: *bathtub, bookshelf, guitar, range hood, sofa, wardrobe, and tv stand*. The results are shown in Table 12. It shows that the network indeed has learned some general features for pose refinement across different object categories.

## 5 Conclusion

In this work we introduce DeepIM, a novel framework for iterative pose matching using color images only. Given an initial 6D pose estimation of an object, we have designed a new deep neural network to directly output a *relative* pose transformation that improves the pose estimate. The network automatically learns to match object poses during training. We introduce an disentangled pose representation that is also independent of the object size and the coordinate frame of the 3D object model. In this way, the network can even match poses of unseen objects, as shown in our experiments. Our method significantly outperforms state-of-the-art 6D pose estimation methods using color images only and provides performance close to methods that use depth images for pose refinement, such as using the iterative closest point algorithm. Example visualizations of our results on LINEMOD, ModelNet, T-LESS can be found here: https://rse-lab.cs.washington.edu/projects/deepim.

This work opens up various directions for future research. For instance, we expect that a stereo version of DeepIM could further improve pose accuracy. Furthermore, DeepIM indicates that it is possible to produce accurate 6D pose estimates using color images only, enabling the use of cameras that capture high resolution images at high frame rates with a large field of view, providing estimates useful for applications such as robot manipulation.

**Acknowledgements** We thank Lirui Wang at University of Washington for his contribution in this project. This work was funded in part by a Siemens Grant. We would also like to thank NVIDIA for generously providing the DGX station used for this research via the NVIDIA Robotics Lab and the UW NVIDIA AI Lab (NVAIL). This work was also Supported by National Key R&D Program of China 2017YFB1002202, NSFC Projects 61620106005, 61325003, Beijing Municipal Sci. & Tech. Commission Z181100008918014 and THU Initiative Scientific Research Program.

## References

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, *110*(3), 346–359.

Besl, P. J., & McKay, N. D. (1992). Method for registration of 3-d shapes. In P. J. Besl & N. D. McKay (Eds.), *Sensor fusion IV: Control paradigms and data structures* (Vol. 1611, pp. 586–607). Bellingham: International Society for Optics and Photonics.

Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., & Rother, C. (2014). Learning 6D object pose estimation using 3D object coordinates. In: European conference on computer vision (ECCV).

Brachmann, E., Michel, F., Krull, A., Ying Yang, M., Gumhold, S., & Rother, C. (2016). Uncertainty-driven 6D pose estimation of objects and scenes from a single RGB image. In: IEEE conference on computer vision and pattern recognition (CVPR) (pp. 3364–3372).

Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., & Dollar, A. M. (2015). The YCB object and model set: Towards common benchmarks for manipulation research. In: 2015 International conference on advanced robotics (ICAR), IEEE (pp. 510–517).

Carreira, J., Agrawal, P., Fragkiadaki, K., & Malik, J. (2016). Human pose estimation with iterative error feedback. In: IEEE conference on computer vision and pattern recognition (CVPR).

Collet, A., Martinez, M., & Srinivasa, S. S. (2011). The MOPED framework: Object recognition and pose estimation for manipulation. *International Journal of Robotics Research (IJRR)*, *30*(10), 1284–1306.

Costante, G., & Ciarfuglia, T. A. (2018). LS-VO: Learning dense optical subspace for robust visual odometry estimation. *IEEE Robotics and Automation Letters*, *3*(3), 1735–1742.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, *1*, 886–893.

Deng, X., Mousavian, A., Xiang, Y., Xia, F., Bretl, T., & Fox, D. (2019). PoseRBPF: A Rao-blackwellized particle filter for 6D object pose tracking. In *Robotics: Science and systems (RSS)*.

Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., van der Smagt, P., Cremers, D., & Brox, T. (2015). Flownet: Learning optical flow with convolutional networks. In: *IEEE international conference on computer vision (ICCV)*, pp 2758–2766.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., & Zisserman, A. (2010). The pascal visual object classes (VOC) challenge. *IEEE International Journal of Computer Vision (ICCV)*, *88*(2), 303–338.

Garon, M., & Lalonde, J. F. (2017). Deep 6-DOF tracking. *IEEE Transactions on Visualization and Computer Graphics*, *23*(11), 2410–2418.

Garon, M., Boulet, P. O., Doironz, J. P., Beaulieu, L., & Lalonde, J. F. (2016). Real-time high resolution 3D data on the hololens. In *IEEE international symposium on mixed and augmented reality (ISMAR-Adjunct)*, IEEE (pp. 189–191).

Girshick, R. (2015). Fast R-CNN. In: *IEEE international conference on computer vision (ICCV)* (pp. 1440–1448).

Gu, C., & Ren, X. (2010). Discriminative mixture-of-templates for viewpoint classification. In *European conference on computer vision (ECCV)* (pp. 408–421).

Hinterstoisser, S., Cagniart, C., Ilic, S., Sturm, P., Navab, N., Fua, P., et al. (2012a). Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *34*(5), 876–888.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., & Navab, N. (2012b). Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Asian conference on computer vision (ACCV)*.

Hinterstoisser, S., Lepetit, V., Rajkumar, N., & Konolige, K. (2016). Going further with point pair features. In *European conference on computer vision (ECCV)* (pp. 834–848).

Hodan, T., Haluza, P., Obdržálek, Š., Matas, J., Lourakis, M., & Zabulis, X. (2017). T-less: An RGB-D dataset for 6D pose estimation of texture-less objects. In *IEEE winter conference on applications of computer vision (WACV)*, IEEE (pp. 880–888).

Johnson, A. E., & Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *5*, 433–449.

Jurie, F., & Dhome, M. (2001). Real time 3D template matching. In *IEEE conference on computer vision and pattern recognition (CVPR)* (Vol. 1, p. I).

Kehl, W., Manhardt, F., Tombari, F., Ilic, S., & Navab, N. (2017). SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1521–1529).

Kendall, A., & Cipolla, R. (2017). Geometric loss functions for camera pose regression with deep learning. In *IEEE conference on computer vision and pattern recognition (CVPR)*.

Krull, A., Brachmann, E., Michel, F., Ying Yang, M., Gumhold, S., & Rother, C. (2015). Learning analysis-by-synthesis for 6D pose estimation in RGB-D images. In *IEEE international conference on computer vision (ICCV)* (pp. 954–962).

Lin, C. H., & Lucey, S. (2017). Inverse compositional spatial transformer networks. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 2568–2576).

Liu, M. Y., Tuzel, O., Veeraraghavan, A., & Chellappa, R. (2010). Fast directional chamfer matching. In: *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1696–1703).

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In *European conference on computer vision (ECCV)* (pp. 21–37).

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 3431–3440).

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *IEEE international conference on computer vision (ICCV)* (Vol. 2, pp. 1150–1157).

Manhardt, F., Kehl, W., Navab, N., & Tombari, F. (2018). Deep model-based 6D pose refinement in RGB. In *European conference on computer vision (ECCV)* (pp. 800–815).

Mellado, N., Aiger, D., & Mitra, N. J. (2014). Super 4pcs fast global pointcloud registration via smart indexing. *Computer Graphics Forum*, *33*, 205–215.

Mian, A. S., Bennamoun, M., & Owens, R. (2006). Three-dimensional model-based object recognition and segmentation in cluttered scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *28*(10), 1584–1601.

Michel, F., Kirillov, A., Brachmann, E., Krull, A., Gumhold, S., Savchynskyy, B., & Rother, C. (2017). Global hypothesis generation for 6D object pose estimation. In *IEEE conference on computer vision and pattern recognition (CVPR)*.

Mousavian, A., Anguelov, D., Flynn, J., & Košecká, J. (2017). 3D bounding box estimation using deep learning and geometry. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 5632–5640).

Nistér, D. (2005). Preemptive RANSAC for live structure and motion estimation. *Machine Vision and Applications*, *16*(5), 321–329.

Oberweger, M., Wohlhart, P., & Lepetit, V. (2015). Training a feedback loop for hand pose estimation. In *IEEE international conference on computer vision (ICCV)*.

Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3D classification and segmentation. *IEEE Computer Vision and Pattern Recognition (CVPR)*, *1*(2), 4.

Rad, M., & Lepetit, V. (2017). BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth. In *IEEE international conference on computer vision (ICCV)*.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 779–788).

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems (NIPS)*.

Rothganger, F., Lazebnik, S., Schmid, C., & Ponce, J. (2006). 3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision (IJCV)*, *66*(3), 231–259.

Rusinkiewicz, S., & Levoy, M. (2001). Efficient variants of the ICP algorithm. In: Third international conference on 3-D digital imaging and modeling, 2001. Proceedings. IEEE (pp. 145–152).

Rusu, R. B., Blodow, N., & Beetz, M. (2009). Fast point feature histograms (FPFH) for 3D registration. In *IEEE international conference on robotics and automation (ICRA)*, Citeseer (pp. 3212–3217).

Salvi, J., Matabosch, C., Fofi, D., & Forest, J. (2007). A review of recent range image registration methods with accuracy evaluation. *Image and Vision Computing*, *25*(5), 578–596.

Saxena, A., Pandya, H., Kumar, G., Gaud, A., & Krishna, K. M. (2017). Exploring convolutional networks for end-to-end visual servoing. In *IEEE international conference on robotics and automation (ICRA)* (pp. 3817–3823).

Shotton, J., Glocker, B., Zach, C., Izadi, S., Criminisi, A., & Fitzgibbon, A. (2013). Scene coordinate regression forests for camera relocalization in RGB-D images. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 2930–2937).

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

Sundermeyer, M., Marton, Z. C., Durner, M., Brucker, M., & Triebel, R. (2018). Implicit 3D orientation learning for 6D object detection from RGB images. In *European conference on computer vision (ECCV)* (pp. 699–715).

Tam, G. K., Cheng, Z. Q., Lai, Y. K., Langbein, F. C., Liu, Y., Marshall, D., et al. (2013). Registration of 3D point clouds and meshes: A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics*, *19*(7), 1199–1217.

Tekin, B., Sinha, S. N., & Fua, P. (2017). Real-time seamless single shot 6D object pose prediction. arXiv preprint arXiv:1711.08848.

Theiler, P. W., Wegner, J. D., & Schindler, K. (2015). Globally consistent registration of terrestrial laser scans via graph optimization. *ISPRS Journal of Photogrammetry and Remote Sensing*, *109*, 126–138.

Tjaden, H., Schwanecke, U., & Schömer, E. (2017). Real-time monocular pose estimation of 3D objects using temporally consistent local color histograms. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 124–132).

Tombari, F., Salti, S., & Di Stefano, L. (2010). Unique signatures of histograms for local surface description. In *European conference on computer vision (ECCV)*, Springer (pp. 356–369).

Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., & Birchfield, S. (2018). Deep object pose estimation for semantic robotic grasping of household objects. In *Conference on robot learning* (pp. 306–316).

Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., & Savarese, S. (2019). Densefusion: 6D object pose estimation by iterative dense fusion. arXiv preprint arXiv:1901.04780.

Wang, S., Clark, R., Wen, H., & Trigoni, N. (2017). Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *IEEE international conference on robotics and automation (ICRA)*, IEEE (pp. 2043–2050).

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3D shapenets: A deep representation for volumetric shapes. In *IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 1912–1920).

Xiang, Y., Schmidt, T., Narayanan, V., & Fox, D. (2018). PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. In *Robotics: Science and systems (RSS)*.

Yang, J., Li, H., Campbell, D., & Jia, Y. (2016). GO-ICP: a globally optimal solution to 3D ICP point-set registration. arXiv preprint arXiv:1605.03344.

Zeng, A., Yu, K. T., Song, S., Suo, D., Walker, E., Rodriguez, A., & Xiao, J. (2017). Multi-view self-supervised deep learning for 6D pose estimation in the Amazon picking challenge. In *IEEE international conference on robotics and automation (ICRA)* (pp. 1386–1383).

Zhou, Q. Y., Park, J., & Koltun, V. (2016). Fast global registration. In *European conference on computer vision (ECCV)*, Springer (pp. 766–782).