

MaskedFusion: Mask-based 6D Object Pose Estimation

Nuno Pereira^[0000–0001–7177–751X] and Luís A. Alexandre^[0000–0002–5133–5025]

NOVA LINCS, Universidade da Beira Interior, Covilhã, Portugal
 {nuno.pereira,luis.alexandre}@ubi.pt

Abstract. MaskedFusion is a framework to estimate the 6D pose of objects using RGB-D data, with an architecture that leverages multiple sub-tasks in a pipeline to achieve accurate 6D poses. 6D pose estimation is an open challenge due to complex world objects and many possible problems when capturing data from the real world, *e.g.*, occlusions, truncations, and noise in the data. Achieving accurate 6D poses will improve results in other open problems like robot grasping or positioning objects in augmented reality. MaskedFusion improves the state-of-the-art by using object masks to eliminate non-relevant data. With the inclusion of the masks on the neural network that estimates the 6D pose of an object we also have features that represent the object shape. MaskedFusion is a modular pipeline where each sub-task can have different methods that achieve the objective. MaskedFusion achieved 97.3% on average using the ADD metric on the LineMOD dataset and 93.3% using the ADD-S AUC metric on YCB-Video Dataset, which is an improvement, compared to the state-of-the-art methods. The code is available on GitHub (<https://github.com/kroglice/MaskedFusion>).

1 Introduction

Object detection and pose estimation are important tasks for robotic manipulation, scene understanding and augmented reality, and it has recently been the target of much research effort. With the increasing automation and the need for robots that can work in non-restricted environments, the capacity to understand the scene in 3 dimensions is becoming a must. One of the main tasks involving robots is grasping objects. Performing grasping in a non-restricted or/and cluttered environment, *e.g.*, bin picking, is a hard problem to tackle. 6D pose estimation is a task in computer vision that detects the 6D pose (3 degrees of freedom for the position and the other 3 for orientation) of an object. 6D pose estimation is an open important problem because it can be used in several important tasks like grasping, robotic manipulation, augmented reality and others. 6D pose is as important in robotic tasks as in augmented reality, where the pose of real objects can affect the interpretation of the scene and the pose of virtual objects can also improve the augmented reality experience. It can also be useful in human-robot interaction tasks such as learning from demonstration and human-robot collaboration. Estimating the object’s 6D pose is a challenging problem due to the

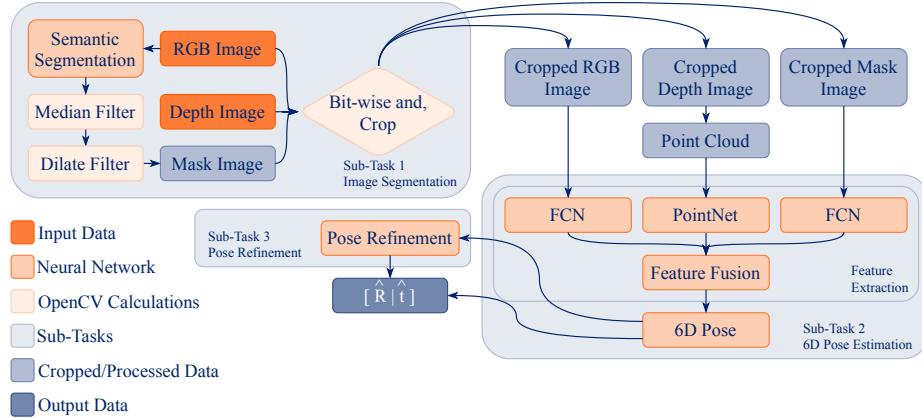


Fig. 1: Pipeline diagram of the MaskedFusion architecture. MaskedFusion has three sub-tasks: image segmentation, 6D pose estimation, and a pose refinement

different objects that exist and how they appear in the real world. Captured scenes from the real world might have occlusions and truncations on some objects. Obtaining the data to retrieve the 6D pose is also a problem, as RGB-D data can be hard to obtain for certain types of object, *e.g.*, fully metallic objects and meshed office garbage bins. 6D pose estimation can be split into three different categories, defined by the type of input data that the methods use. Methods [2], [15], [16], [17], [23], [25], [26] that use RGB images as input usually rely on the detection and matching of keypoints from the objects in a scene with the 3D render and use the *PnP* [6] algorithm to solve the pose of that object. Point cloud methods [13], [18], [19], [21], [22], [33] rely on descriptors to extract features from the objects in scene to later be matched with features captured in known poses. Finally, RGB-D methods [8], [9], [12], [28], [30], [31] have the 6D poses directly regressed from the data, and then furthered refined. Most of the methods in this category only use the depth data in the refinement phase. Our method, MaskedFusion, fits in the RGB-D category because we directly regress the 6D pose from the RGB-D data.

We propose a new method based on a pipeline with 3 sub-tasks that, when combined, can estimate the object's 6D pose. For the first sub-task, we detect and classify each object in the scene using semantic segmentation and retrieve their binary masks. Then, in the second sub-task, for each object detected, we extract features from the different types of data and fuse them in a pixel-wise manner. After the fusion, we have the 6D pose neural network (NN) that will regress the 6D pose of the object. The third sub-task is optional but advisable and refines the 6D pose of the object. For the refinement sub-task, we used a NN that was proposed in [28].

MaskedFusion improves upon the state-of-the-art method by achieving greater scores in two (LineMOD [7], YCB-Video [30]) of the most used datasets in the 6D pose estimation area. Our method achieved an average of 97.3% and our best

experience achieved 97.8% in the LineMOD dataset using the ADD (2) metric. For the YCB-Video dataset, we achieved on average 93.3% using the ADD-S (3) AUC metric and 97.1% of ADD-S smaller than $2cm$.

In summary, our work has the following main contributions:

- End-to-end modular pipeline to estimate the object’s 6D pose;
- A new neural network to estimate 6D pose that can be used outside of our pipeline, it is also very fast to execute during inference;
- Improvements to the state-of-the-art on the two most used 6D pose evaluation datasets.

2 Related Work

In this section, we present the most relevant literature. In subsection 2.1 we present methods of semantic segmentation that use CNNs and FCNs. On the following subsection 2.2 we present the relevant literature for 6D pose estimation.

2.1 Semantic Segmentation

The semantic segmentation goal is to classify each pixel in an image. In contrast, the instance segmentation goal is to detect, delineate it with a bounding box and segment or create a mask for each object instance present in the image. The objective of the panoptic segmentation task [10] is the unification of the two typically distinct tasks of semantic segmentation and instance segmentation for a given image.

Some of the most notable techniques used in semantic segmentation are presented in this section starting with the U-Net [20]. It is most known for its U-shaped architecture that does semantic segmentation with the encoder-decoder method. The authors of U-net created its architecture with two consecutive convolutions followed by a max-pooling. With this method, the spatial information is decreased but feature information is increased.

U-Net and SegNet [1] have similar architectures where the second half of those architectures consists of the same structure in the first half but hierarchically opposite. SegNet is a FCN based on the 13 VGG-16 [11] convolutional layers. Another technique commonly found in semantic segmentation is addressing it using contextual information.

Pyramid Scene Parsing Network (PSPNet) [32], also uses global contextual information for semantic segmentation. The authors introduced a Pyramid Pooling Module after the last layers of an FCN (based on ResNet-18), the feature maps obtained from the FCN are pooled using 4 different scales corresponding to 4 different pyramid levels each with different sizes, 1×1 , 2×2 , 3×3 and 6×6 . The polled feature maps are then convoluted using a 1×1 convolution layer to reduce the feature maps dimension. So the outputs of each convolution are up-sampled and then concatenated with the initial feature maps that were obtained in the FCN. This concatenation provides the local and global contextual information of the image. After the concatenation, the authors use another convolution layer to

generate the final pixel-wise predictions. The main idea of PSPNet is to observe the whole feature map in sub-regions with different locations using the pyramid pooling module, such that, the network achieves a better understanding of the scene.

2.2 6D Pose Estimation

Methods that estimate object 6D pose can be split into three different categories, RGB, Point Cloud, RGB-D. These categories are defined by the type of input data that methods use.

Methods that use RGB images as input [2], [15], [16], [17], [23], [25], [26] usually rely on the detection and matching of keypoints from the objects in a scene with the 3D render and use the *PnP* [6] algorithm to solve the pose of that object. Although they are very fast methods, their recall drops quickly in the presence of occlusion. The methods that do not use this technique are methods that use deep learning [15], [23]. These methods rely on convolutional neural networks (CNNs) to extract and store features from the multiple viewpoint. This process is done during the training phase of the CNN. On the inference phase, features are extracted from the new scenes and then matched with previously known features and with this matching process it is possible to obtain the object 6D pose.

Wu *et al.* [29] method is made to be fast, running at 20Hz on RGB images with accurate 6D object poses achieved from segmenting the RGB image to obtain the masks and the classes of the objects and then for each mask, with the use of a pose interpreter network, it regresses the 6D pose of the object. The pose interpreter network learns with a new proposed loss similar to L_1 but applied on the point clouds generated from the objects. One of the most accurate method in 6D pose using RGB images is PVNet [17]. PVNet is a hybrid method that uses the two methods, segmentation and then *PnP*. With this combination, it can handle mild occlusion and truncations. PVNet has a NN to predict pixel-wise object labels and unit vectors that represent the direction from each pixel to every keypoint of the object. These keypoints are voted and matched with the 3D model of the object to estimate its 6D pose.

Point cloud methods [13], [18], [19], [21], [22], [33] rely on descriptors to extract features from the objects in the scene to later be matched with features captured in known poses. Methods like PVFH [13] and its predecessors [21], [22] achieve remarkable speed acquiring the 6D pose of the object, but these methods need good data retrieval to be accurate and most of the data captured from the real world usually has different types of interference and/or noise. So deep learning is also contributing in this category of methods to achieve better accuracy and get better extracted features from the objects, mitigating some of the noise and interference that can appear. Many deep learning architectures were propose, like PointNets [18], [19] and VoxelNet [33]. These methods achieved good results in multiple datasets.

Finally, RGB-D methods [8], [9], [12], [28], [30], [31] usually have the object 6D pose directly regressed from the data, and usually further refined by other

methods, *e.g.*, Iterative Closest Point (ICP). Tejani *et al.* [24] follow a local approach where small RGB-D patches vote for object pose hypotheses in a 6D space. Kehl *et al.* [9] also follow a local approach but they use a convolutional auto-encoder (CAE) to encode each patch of the object to later match these features obtained in the bottleneck of the CAE with a code-book of features learned during the train, and use the code-book matches to predict the 6D pose of the object. Although such methods are not taking global context into account, they proved to be robust to occlusion and the presence of noise artifacts since they infer the object pose using only small patches of the image. SSD-6D [8] uses an RGB image that is processed by the NN to output localized 2D detection with bounding boxes and then it classifies the bounding boxes into discrete bins of Euler angles and subsequently estimates the object 6D pose. This method is in the RGB-D category because after the first estimation, and with the availability of the depth information, the 6D poses can be further refined. PoseCNN [30] uses a new loss function that is robust to object symmetry to directly regress the object rotation. It uses a Hough voting approach to obtain the 3D location center of the object to achieve its translation. Using ICP on the refinement phase of SSD-6D and PoseCNN makes their 6D pose estimation accurate. Li *et al.* [12] formulates a discriminative representation of 6-D pose that enables predictions of both rotation and translation by a single forward pass of a CNN, and it can be used with many object categories. DenseFusion [28] extract features from RGB images and depth data with different FCN. After the extraction, it fuses the depth feature with the RGB features while retaining the input’s space geometric structure, and then it estimates the object 6D pose. DenseFusion is similar to PointFusion [31], as it also estimates the 6D pose while keeping the geometric structure and appearance information of the object, to later fuse this information in a heterogeneous architecture.

3 Methodology

Our method, MaskedFusion, is a pipeline (Fig. 1) dived into 3 sub-tasks that combined can solve the task of object 6D pose estimation. MaskedFusion is a modular pipeline, for each sub-task we use a NN to solve the task that it is responsible for. However, since our pipeline is modular every sub-task can have different types of methods that will solve the task in hand and they can be replaced easily.

In the first sub-task, is where we need to detect and segment each object in the scene. To do that we use a NN based on the encoder-decoder architecture to classify each pixel of the RGB image captured, and predict the mask and the location for each object in the scene.

For the second sub-task, with the masks obtained in sub-task 1 for each object and the RGB-D data, we can now estimate the object 6D pose. After this sub-task, we obtain the rotation matrix and the translation vector according to the 6D pose estimated from the 6D pose NN. After obtaining the translation vector

and rotation matrix we can use another method to refine further the estimated 6D pose. This last sub-task is optional but advisable for better results.

In the last sub-task, we use the same refinement network as the DenseFusion [28]. This last step can be slow during the training because it needs many training epochs to show its advantages, but during the inference, it is very fast. Other methods usually use ICP that is also a good method for refinement but it has a higher computational cost and will take longer to apply the refinements during the inference time.

During the training phase of our pipeline, both of the main NNs, semantic segmentation and 6D pose, are trained independently.

3.1 Semantic Segmentation

Semantic labeling provides richer information about the objects and handles occlusions better than object detection and localization methods. For the first sub-task of our pipeline, a semantic segmentation method is used to detect and extract the mask of each object presented in the scene. For the completion of this sub-task, we only use the RGB image.

For the semantic segmentation, we use an FCN with encoder-decoder architecture. Our implementation is similar to Segnet [1], however, since our pipeline is modular it is possible to use other methods to detect and obtain the object masks. We used and tested with a semantic segmentation method but it is also possible to use an instance segmentation, panoptic segmentation, or any other methods that can detect and output the mask for each object in the scene.

In our implementation of the semantic segmentation, after the output of the mask, we apply 2 filters on the mask before saving or feed-forward it to the next sub-task. First, we use the median filter from *OpenCV* to smooth the image with a kernel size of 3×3 . Second, we dilate the mask with a 5×5 kernel such that, if the mask has some minor boundary segmentation error, this operation helps to correct it.

The binary mask obtained from the semantic segmentation method with the 2 filters applied is used to crop the RGB and depth images. This is done for each object present in the scene. To crop the RGB and depth images, we apply a *bit-wise and* between the RGB image and mask, and also between the depth image and the mask. The result of the *bit-wise and* of the RGB image and the mask will be inside of a rectangular crop that encloses all the object and this smaller image will serve as input data to the 6D pose NN. In the case of the depth image, a point cloud is further generated from the resultant *bit-wise and* cropped depth image, and the point cloud will also serve as input to the next phase of our pipeline. Cropping the data with the mask is a pre-processing of the data that helps the 6D pose NN on the second sub-task because it discards the background or other non-relevant data that are around the object leaving only the data that is most relevant to the 6D pose estimation.

3.2 6D Pose Neural Network

Our NN that estimates the 6D pose of the object is inspired in DenseFusion [28] and PointFusion [31]. It has a similar architecture that is used to extract features from different types of data and fuse the information in a pixel-wise manner. Our method improves upon DenseFusion and others, and we show our performance in the results section (5). Fig. 1 shows the architecture of our 6D pose NN (Sub-task 2). Our 6D pose NN can be divided into two stages: feature extraction and 6D pose estimation.

On the first stage, feature extraction, all the data that were cropped after the semantic segmentation are separated into different NNs that extract features for each type of data. For the point cloud data that were generated from the cropped depth image, we used the PointNet [19] to extract 500 features that represent the depth information of the object. For the cropped RGB image we use a fully convolutional neural network (FCN) based on *ResNet-18* to also extract 500 features from the color image. Our *ResNet-18* is similar to the original but without the fully connected layers at the end making it an FCN. Finally, for the binary mask image, we also used an FCN similar to the FCN used for the color images, but in this case, at the start, we only have one channel as input instead of three. As in the color images for the masks, we extract 500 features. Having features from the mask enables us to have features that represent the shape of the object which is more relevant information for our next stage and improved the accuracy of our method.

On the second stage, all the extracted features of each data source are then concatenated into a single vector and pass through a convolutional layer to fuse all the features. We then use another NN that receives the features extracted previously and regresses the 6D pose of the object, that is, the rotation matrix and the translation vector of the object. With all the features concatenated from each input, and then feed-forward to a convolutional layer, we will have 2 different paths (each with 4 convolutional layers), one of the paths is for the regression of the translation vector and the other is to regress the rotation matrix.

So that our NN can regress the 6D pose we use (1) as loss function during the train. This loss function is the same that the DenseFusion used:

$$\mathcal{L}_i^p = \frac{1}{M} \sum_j \left\| (Rx_j + t) - (\hat{R}_i x_j + \hat{t}_i) \right\| \quad (1)$$

where, x_j denotes the j^{th} point of the M randomly selected 3D points from the objects 3D model, $p = [R|t]$ is the ground truth pose, where R is the rotation matrix of the object and t is the translation. The predicted pose generated from the fused embedding of the i^{th} dense-pixel is represented by $\hat{p}_i = [\hat{R}_i|\hat{t}_i]$ where \hat{R} denotes the predicted rotation and \hat{t} the predicted translation. After training the 6D pose NN, the output of it ($\hat{p}_i = [\hat{R}_i|\hat{t}_i]$) can be retrieved after the second stage or it can be sent to the next sub-task of the pipeline.

3.3 Pose Refinement

In the last sub-task, we used the same pose refinement that DenseFusion developed. The authors of DenseFusion created a pose refinement NN that improves upon the 6D pose previously estimated. The output 6D poses estimated before serves as input to the DenseFusion pose refinement NN.

We tested the DenseFusion refinement NN and concluded that it is very slow during the training because it needs many training epochs to start showing its advantages, but during the inference, it is very fast. Other refinement methods, can be used but they usually require more computation during the inference and it slows down the process of estimating the 6D pose.

4 Experiments

We tested our method on two widely-used datasets, LineMOD [7] and YCB-Video [30]. We use these datasets because it is easier to compare our method with previous methods that were also tested in the same datasets.

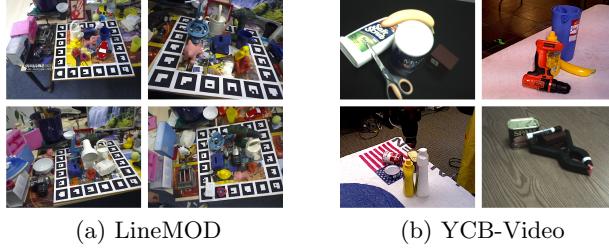


Fig. 2: Example of images from the datasets

For each experiment we trained the methods from scratch and then evaluate them. We repeat this procedure 5 times and present the average results, whereas other references usually only present one result. All experiments were executed on a desktop with SSD NVME, 64GB of RAM, an NVIDIA GeForce GTX 1080 Ti and Intel Core i7-7700K CPU.

4.1 LineMOD

LineMOD [7] is one of the most used datasets to tackle the 6D pose estimation problem. Many types of methods that tackle the 6D pose estimation problem use this dataset ranging from the classical methods like [3], [5], [27] to the most recent deep learning approaches [14], [28], [30]. This dataset was captured with a Kinect, and it has a procedure that automatically aligns the RGB and depth images. LineMOD has 15 low-textured objects (although we only use 13 as in previous methods) in over 18000 real images and has the ground truth pose annotated.

Each object is associated with a test image showing one annotated object instance with significant clutter but only mild occlusion, as shown in Fig. 2.

Each object also contains the 3D model saved as a point cloud and a distance file with the maximum diameter (*cm*) of the object.

4.2 YCB-Video

After the release of the YCB-Video dataset in [30] it started being used by methods that can estimate the 6D pose of objects even in scenes with heavy occlusions. This dataset has 21 objects that were presented in [4] by Calli *et al.* These objects have different shapes and textures, and mild occlusions. YCB-Video is composed of 92 RGB-D videos, each with a subset of the objects placed in the scene. These videos have the segmentation masks and the 6D poses of the objects annotated, and for each frame there are at least 3 objects present in the scene. We have used the dataset in the same splits has previous works, [30], [28], where 80 videos were used for training and 12 for testing. The 80,000 synthetic images released in [30] were also used during the train of our 6D pose sub-task, but not used for the semantic segmentation sub-task.

4.3 Metrics

As in previous works [8], [17], [28], [30] we used the Average Distance of Model Points (ADD) [7] as metric of evaluation for non-symmetric objects and for the egg-box and glue we used the Average Closest Point Distance (ADD-S) [30]. We needed to use another metric for these two objects because they are symmetric objects.

$$\text{ADD} = \frac{1}{m} \sum_{x \in M} \left\| (Rx + t) - (\hat{R}x + \hat{t}) \right\| \quad (2)$$

In the ADD metric (equation 2), assuming the ground truth rotation R and translation t and the estimated rotation \hat{R} and translation \hat{t} , the average distance calculates the mean of the pairwise distances between the 3D model points of the ground truth pose and the estimated pose. In equation (2) and (3) M represents the set of 3D model points and m is the number of points.

For the symmetric objects (egg-box and glue), the matching between points is ambiguous for some poses. In these cases we used the ADD-S metric:

$$\text{ADD-S} = \frac{1}{m} \sum_{x_1 \in M} \min_{x_2 \in M} \left\| (Rx_1 + t) - (\hat{R}x_2 + \hat{t}) \right\| \quad (3)$$

For the YCB-Video we also used the same metrics as in previous works like PoseCNN [30] and DenseFusion [28]. We use the area under the ADD-S (3) curve (AUC) as in DenseFusion [28] where their threshold was 10*cm* the same as PoseCNN [30], and we also calculate the percentage of ADD-S smaller than 2*cm*, which DenseFusion [28] authors consider as the minimum tolerance for most of the robot grippers. Using these same metrics in both datasets as previous works enable us to make direct comparisons with their methods.

5 Results

In this section, we present the results of the experiments made and compare them with other methods that have been evaluated in the same datasets. The inference results in subsection 5.4 present the results for the full pipeline, meaning that we do not use the masks provided in the YCB-Video dataset, but instead rely on the masks produced by the segmentation sub-task of MaskedFusion. This makes the task harder since the masks provided with YCB-Video are all correct whereas the ones we receive from the segmentation can have errors.

5.1 Results in LineMOD

We present here the results of the NN that estimates the object 6D pose of the MaskedFusion pipeline using the LineMOD dataset.

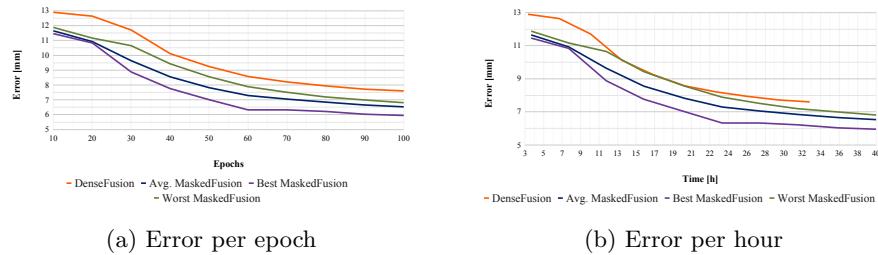


Fig. 3: The methods were evaluated on the test set, after every 10 training epochs and the figure contains the average error in millimeters. Figure (a) shows the error as a function of the training epoch whereas figure (b) presents it as a function of training time. All MaskedFusion runs got smaller average error than DenseFusion

In Fig. 3 (a), we show the test results for several different epochs using the LineMOD dataset. We trained both MaskedFusion and the DenseFusion for 100 epochs, and every 10 epochs we tested them and plotted their mean errors. It can be seen in Fig. 3 (a) that even our worst values are better than the best values of DenseFusion. All MaskedFusion average error values were always below the DenseFusion in all epochs tested, and most important is that our method entered first in the 10mm error mark. Our method achieves the 10mm error, on average, in epoch 30 and DenseFusion only achieved this error in epoch 40. Comparing the mean error of our best run to the DenseFusion best result we have an error of 5.9mm and DenseFusion has 7.6mm. We achieved more accuracy leading to possible better placement of objects in a virtual scene or better grasping accuracy. To train our method for 100 epochs took 40 hours, compared with 33 for DenseFusion. Since we have one more network to train and we have additional computation over the data and more data flowing in our method, it is normal to take longer in the overall training. Note that, since

Table 1: Quantitative evaluation of 6D pose using the ADD metric on the LineMOD dataset. Symmetric objects are presented in italic and were evaluated using ADD-S. Bold shows best results in a given row

Objects	SSD-6D + ICP	PointFusion	DenseFusion	MaskedFusion		MaskedFusion				
				Avg	Stdev	Individual Experiments				
ape	65.0	70.4	92.3	92.2	4.1	97.1	86.7	90.5	91.4	95.2
bench vi.	80.0	80.7	93.2	98.4	1.1	98.1	100.0	97.1	99.0	98.1
camera	78.0	60.8	94.4	98.0	1.0	97.1	97.1	99.0	99.0	98.0
can	86.0	61.1	93.1	97.4	2.3	99.0	98.0	95.0	100.0	95.0
cat	70.0	79.1	96.5	97.8	1.5	96.0	98.0	97.0	100.0	98.0
driller	73.0	47.3	87.0	95.6	1.7	95.0	96.0	93.0	97.0	97.0
duck	66.0	63.0	92.3	94.0	3.0	97.2	89.6	95.3	92.5	95.3
<i>eggbox</i>	100.0	99.9	99.8	99.6	0.5	100.0	100.0	99.1	99.1	100.0
<i>glue</i>	100.0	99.3	100.0	100.0	0.0	100.0	100.0	100.0	100.0	100.0
hole p.	49.0	71.8	92.1	97.3	2.5	93.3	97.1	98.1	100.0	98.1
iron	78.0	83.2	97.0	97.1	1.3	95.9	99.0	97.9	96.9	95.9
lamp	73.0	62.3	95.3	99.0	1.0	98.1	100.0	100.0	98.1	99.0
phone	79.0	78.8	92.8	98.8	1.3	97.1	100.0	100.0	99.0	98.1
Average	76.7	73.7	94.3	97.3	0.3	97.2	97.0	97.1	97.8	97.5

we can achieve a smaller error before DenseFusion in terms of training epochs, this increase in training time can be disregarded, since, even if the training was stopped after a fixed number of hours instead of after a fixed number of epochs, MaskFusion would still produce a smaller estimation error, as can be seen in Fig. 3 (b). For instance, MaskedFusion entered in the 10mm error mark after 12 hours of training while DenseFusion needed 13.2 hours.

In Table 1 we present a comparison of our test results in a per object comparison with other three methods, SSD-6D [8], PointFusion [31] and DenseFusion [28]. The values presented in Table 1 result from the ADD metric (equation 2) and ADD-S metric (equation 3). From Table 1, we conclude that our method has overall better accuracy than previous methods. The average results from the 5 repetitions of our method are better than DenseFusion in 11 out of 13 objects. In our worst-performing experience, the second column of MaskedFusion Individual Experiments in Table 1, we achieved an average of 97%, which is better than the DenseFusion. Finally, the best of our 5 repetitions improves the overall ADD from the 94.3% of DenseFusion to 97.8%. For the LineMOD dataset, we have achieved overall better results than previous methods.

5.2 Results in YCB-Video

We also use the YCB-Video dataset to evaluate our method and compare it with other methods. The results presented in Table 2 show the area under the accuracy-threshold curve (AUC) using the ADD-S metric with the maximum threshold of 10cm.

As shown in Table 2, MaskedFusion obtained the best average score using 200 epochs for training. Since DenseFusion, that was the closest one to us,

Table 2: Quantitative evaluation of 6D pose (area under the ADD-S (3) curve (AUC)) on the YCB-Video Dataset. Bold numbers are the best in a row and underline numbers are the best when comparing MaskedFusion with DenseFusion both with 100 training epochs. The last column of the table is the evaluation of MaskedFusion using the masks that were generated by our first sub-task during the train and test. (*) The values presented were obtained from [28]

Training Epochs Objects	PointFusion	PoseCNN +ICP	DenseFusion	200		100		100		Pipeline
	-	-	-	AUC	AUC	AUC	AUC	AUC	AUC	100
		AUC(*)	AUC(*)	AUC(*)	(Avg)	(Stdev)	(Avg)	(Stdev)	(Avg)	(Stdev)
002_master_chef_can	90.9	95.8	96.4	95.5	0.1	<u>95.9</u>	0.1	94.3	0.7	95.0
003_cracker_box	80.5	92.7	95.5	96.7	0.4	<u>96.0</u>	0.4	94.0	0.7	96.6
004_sugar_box	90.4	98.2	97.5	98.1	0.2	<u>97.6</u>	0.2	95.7	1.6	98.2
005_tomato_soup_can	91.9	94.5	94.6	94.3	0.1	<u>94.2</u>	0.1	90.3	4.0	94.7
006_mustard_bottle	88.5	98.6	97.2	98.0	0.2	<u>97.6</u>	0.2	95.2	1.8	98.0
007_tuna_fish_can	93.8	97.1	96.6	96.9	0.3	<u>96.7</u>	0.3	95.4	0.4	96.8
008_pudding_box	87.5	97.9	96.5	97.3	0.5	<u>96.3</u>	0.5	95.1	0.8	98.2
009_gelatin_box	95.0	98.8	98.1	98.3	0.6	<u>98.0</u>	0.6	97.2	0.5	98.8
010_potted_meat_can	86.4	92.7	91.3	89.6	0.2	<u>89.4</u>	0.2	88.1	0.5	91.0
011_banana	84.7	97.1	96.6	97.6	0.1	<u>97.5</u>	0.1	95.0	0.5	97.3
019_pitcher_base	85.5	97.8	97.1	97.7	0.3	<u>97.4</u>	0.3	96.1	0.5	97.6
021_bleach_cleanser	81.0	96.9	95.8	95.4	0.8	93.8	0.8	<u>94.6</u>	0.1	96.1
024_bowl	75.7	81.0	88.2	89.6	3.1	<u>90.1</u>	3.1	88.4	0.2	89.7
025_mug	94.2	95.0	97.1	97.1	0.2	<u>97.0</u>	0.2	95.8	0.5	97.2
035_power_drill	71.5	98.2	96.0	96.7	0.3	<u>96.4</u>	0.3	93.9	1.1	96.6
036_wood_block	68.1	87.6	89.7	91.8	1.2	<u>90.6</u>	1.2	90.2	0.8	90.7
037_scissors	76.7	91.7	95.2	92.7	0.6	<u>93.2</u>	0.6	92.4	1.3	90.6
040_large_marker	87.9	97.2	97.5	97.5	0.3	<u>97.0</u>	0.3	95.7	0.5	97.1
051_large_clamp	65.9	75.2	72.9	71.9	1.3	<u>72.1</u>	1.3	69.7	0.4	74.7
052_extra_large_clamp	60.4	64.4	69.8	71.4	1.0	<u>69.6</u>	1.0	64.5	0.6	58.8
061_foam_brick	91.8	97.2	92.5	94.3	1.1	<u>94.2</u>	1.1	92.0	2.0	93.7
Average	83.9	93.0	93.1	93.3	0.6	<u>92.9</u>	0.6	91.1	0.9	92.7

didn't state how many epochs they used, we also present a comparison between MaskedFusion and DenseFusion using 100 epochs for training. All of our experiments have 5 repetitions so we can compute the average of all executions, this also includes the experiment that we have done on DenseFusion with 100 training epochs. MaskedFusion 6D pose NN achieved an average of more 0.2% in the AUC than what DenseFusion presented. But when we compare the average of our method using 100 training epochs and DenseFusion we achieved an improvement of 1.8% AUC score.

On Table 3 we present the percentage of ADD-S smaller than $2cm$. This metric was introduced in [28] since predictions under $2cm$ are the minimum tolerance for robot manipulation.

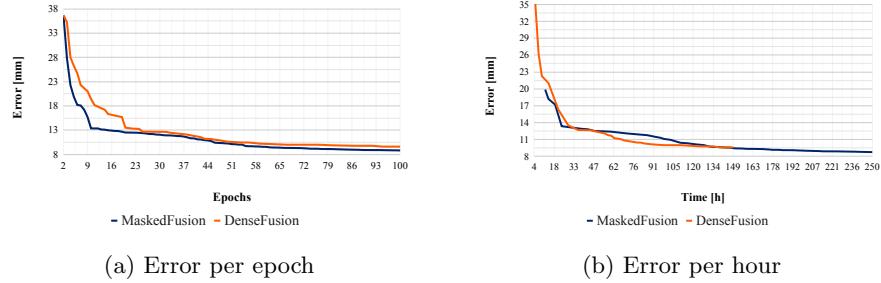
As in the AUC, DenseFusion did not state how many epochs were used for training, so for this metric, we also did the same as in AUC. We trained MaskedFusion and DenseFusion 100 epochs to compare their $< 2cm$ metric and we had an average improvement of 0.3%.

Table 3: Quantitative evaluation of 6D pose (percentage of ADD-S smaller than $2cm$) on the YCB-Video Dataset. Bold numbers are the best in a row and underline numbers are the best when comparing MaskedFusion with DenseFusion both with 100 training epochs. The last column of the table is the evaluation of MaskedFusion using the masks that were generated by our first sub-task during the train and test. (*) The values presented were obtained from [28]

Training Epochs Objects	PointFusion	PoseCNN +ICP	DenseFusion	MaskedFusion	MaskedFusion		DenseFusion		Pipeline
	-	-	-	200 <2cm (Avg)	100 <2cm (Avg)	100 <2cm (Avg)	100 <2cm (Avg)	100 <2cm (Avg)	100 <2cm (Avg)
<2cm(*)	<2cm(*)	<2cm(*)	<2cm(*)	<2cm (Stdev)	<2cm (Stdev)	<2cm (Stdev)	<2cm (Stdev)	<2cm (Stdev)	<2cm (Stdev)
002_master_chef_can	99.8	100.0	100.0	100.0 0.0	100.0 0.0	<u>100.0</u> 0.0	100.0 0.0	100.0 0.0	100.0
003_cracker_box	62.6	91.6	99.5	99.8 0.1	<u>99.7</u> 0.1	99.5 0.5	99.3 0.5	99.3 0.5	99.3
004_sugar_box	95.4	100.0	100.0	100.0 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	100.0
005_tomato_soup_can	96.9	96.9	96.9	96.9 0.0	96.5 0.0	92.6 5.9	96.9 5.9	96.9 5.9	96.9
006_mustard_bottle	84.0	100.0	100.0	100.0 0.0	<u>100.0</u> 0.0	99.9 0.1	100.0 0.0	100.0 0.0	100.0
007_tuna_fish_can	99.8	100.0	100.0	99.7 0.1	99.9 0.1	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	100.0
008_pudding_box	96.7	100.0	100.0	100.0 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	100.0
009_gelatin_box	100.0	100.0	100.0	100.0 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	100.0
010_potted_meat_can	88.5	93.6	93.1	94.2 0.4	<u>90.8</u> 0.4	90.6 0.2	92.8 0.2	92.8 0.2	92.8
011_banana	70.5	99.7	100.0	100.0 0.0	<u>100.0</u> 0.0	99.7 0.5	99.7 0.5	99.7 0.5	99.7
019_pitcher_base	79.8	100.0	100.0	100.0 0.3	99.9 0.3	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	99.8
021_bleach_cleanser	65.0	99.4	100.0	99.4 3.9	94.0 3.9	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	99.6
024_bowl	24.1	54.9	98.8	95.4 5.8	95.8 5.8	95.0 4.2	91.6 4.2	91.6 4.2	91.6
025_mug	99.8	99.8	100.0	100.0 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	100.0
035_power_drill	22.8	99.6	98.7	99.5 0.3	<u>99.4</u> 0.3	97.1 1.5	99.6 1.5	99.6 1.5	99.6
036_wood_block	18.2	80.2	94.6	100.0 0.9	<u>98.1</u> 0.9	97.5 1.5	90.8 1.5	90.8 1.5	90.8
037_scissors	35.9	95.6	100.0	99.9 1.1	99.0 1.1	<u>99.4</u> 0.7	92.8 0.7	92.8 0.7	92.8
040_large_marker	80.4	99.7	100.0	99.9 0.1	99.7 0.1	<u>99.8</u> 0.1	99.4 0.1	99.4 0.1	99.4
051_large_clamp	50.0	74.9	79.2	78.7 0.1	<u>77.9</u> 0.1	75.6 1.2	80.8 1.2	80.8 1.2	80.8
052_extra_large_clamp	20.1	48.8	76.3	75.9 2.1	<u>72.0</u> 2.1	68.7 0.7	75.3 0.7	75.3 0.7	75.3
061_foam_brick	100.0	100.0	100.0	100.0 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	<u>100.0</u> 0.0	99.3
Average	74.1	93.2	96.8	97.1 0.7	<u>96.3</u> 0.7	96.0 0.8	96.1 0.8	96.1 0.8	96.1

As in LineMOD we have one more network to train and we have additional computation over the data and more data flowing in our method, so we took

more 95 of hours to train 100 epochs. MaskedFusion took 240 hours compared with 145 hours of training time for DenseFusion. Fig. 5.3 presents these results both in terms of epochs and in terms of time.



5.3 Results in YCB-Video

5.4 Pipeline results in inference time

The inference time of MaskedFusion in the first sub-task (image segmentation) is 0.2 seconds per image. For the second sub-task (6D pose estimation) 0.01 seconds were needed to estimate the pose per given object, and on the third sub-task, the pose refinement took 0.002 seconds. So the total inference time of MaskedFusion is 0.212 seconds from the moment of the RGB-D image capture until it has the estimated pose.

Using the masks that were generated by our first sub-task during the train we achieved 92.7% in the AUC metrics (shown on the last column of Table 2) and 96.1% in the $< 2\text{cm}$ metric (shown on the last column of Table 3).

Using our masks we had worst overall scores as expected, since we were not using corrected masks as those provided with YCB. With this test we have learned that our semantic segmentation needs to be improved to obtain better masks so that our second sub-task can achieve better 6D pose estimation. We note that other papers in the literature do not present the results on the full pipeline and always consider that the method receives perfectly segmented data.

6 Conclusion

Achieving a robust estimation of the 6D pose of objects captured in the real world is still an open challenge. MaskedFusion improved the state-of-the-art in this area using objects' masks retrieved during the semantic segmentation, to identify and localize the object in the RGB image. We achieved an error below 6mm in LineMOD with just 100 training epochs and in YCB-Video, a challenging dataset, we have obtained AUC score of 93.3% and 97.1% in the $< 2\text{cm}$ metric.

The masks are used to remove non-relevant data from the input of the NN and serve as an additional feature to the 6D pose estimation NN. MaskedFusion has low inference time but at the cost of an increase in training time. We saw that this increased training time can sometimes (such as with LineMOD data and during most of YCB-Video training) be disregarded since we still beat the state-of-the-art if the training time stops after a fixed number of hours instead of a fixed number of epochs. As future work, we intend to study the influence of the instance segmentation method on the MaskedFusion results (since the higher the accuracy of the first sub-task, the better results are expected from the second sub-task) and work towards speeding up the training stage of MaskedFusion.

References

1. Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence* **39**(12), 2481–2495 (2017)
2. Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., Rother, C.: Learning 6d object pose estimation using 3d object coordinates. In: European conference on computer vision. pp. 536–551. Springer (2014)
3. Buch, A.G., Kiforenko, L., Kraft, D.: Rotational subgroup voting and pose clustering for robust 3d object recognition. In: 2017 IEEE International Conference on Computer Vision (ICCV). pp. 4137–4145. IEEE (2017)
4. Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., Dollar, A.M.: The ycb object and model set: Towards common benchmarks for manipulation research. In: 2015 international conference on advanced robotics (ICAR). pp. 510–517. IEEE (2015)
5. Drost, B., Ulrich, M., Navab, N., Ilic, S.: Model globally, match locally: Efficient and robust 3d object recognition. In: 2010 IEEE computer society conference on computer vision and pattern recognition. pp. 998–1005. IEEE (2010)
6. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM* **24**(6), 381–395 (1981)
7. Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., Lepetit, V.: Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In: 2011 international conference on computer vision. pp. 858–865. IEEE (2011)
8. Kehl, W., Manhardt, F., Tombari, F., Ilic, S., Navab, N.: Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1521–1529 (2017)
9. Kehl, W., Milletari, F., Tombari, F., Ilic, S., Navab, N.: Deep learning of local rgbd patches for 3d object detection and 6d pose estimation. In: European Conference on Computer Vision. pp. 205–220. Springer (2016)
10. Kirillov, A., He, K., Girshick, R., Rother, C., Dollár, P.: Panoptic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 9404–9413 (2019)
11. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)

12. Li, C., Bai, J., Hager, G.D.: A unified framework for multi-view multi-class object pose estimation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 254–269 (2018)
13. Li, D., Liu, N., Guo, Y., Wang, X., Xu, J.: 3d object recognition and pose estimation for random bin-picking using partition viewpoint feature histograms. *Pattern Recognition Letters* **128**, 148–154 (2019)
14. Li, Y., Wang, G., Ji, X., Xiang, Y., Fox, D.: Deepim: Deep iterative matching for 6d pose estimation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 683–698 (2018)
15. Mousavian, A., Anguelov, D., Flynn, J., Kosecka, J.: 3d bounding box estimation using deep learning and geometry. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7074–7082 (2017)
16. Pavlakos, G., Zhou, X., Chan, A., Derpanis, K.G., Daniilidis, K.: 6-dof object pose from semantic keypoints. In: 2017 IEEE International Conference on Robotics and Automation (ICRA). pp. 2011–2018. IEEE (2017)
17. Peng, S., Liu, Y., Huang, Q., Zhou, X., Bao, H.: Pvnet: Pixel-wise voting network for 6dof pose estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4561–4570 (2019)
18. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 918–927 (2018)
19. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 652–660 (2017)
20. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
21. Rusu, R.B.: Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. Ph.D. thesis, Computer Science department, Technische Universitaet Muenchen, Germany (October 2009)
22. Rusu, R.B., Bradski, G., Thibaux, R., Hsu, J.: Fast 3d recognition and pose using the viewpoint feature histogram. In: Proceedings of the 23rd IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). Taipei, Taiwan (October 2010)
23. Suwajanakorn, S., Snavely, N., Tompson, J.J., Norouzi, M.: Discovery of latent 3d keypoints via end-to-end geometric reasoning. In: Advances in Neural Information Processing Systems. pp. 2059–2070 (2018)
24. Tejani, A., Kouskouridas, R., Doumanoglou, A., Tang, D., Kim, T.: Latent-class hough forests for 6 dof object pose estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **40**(1), 119–132 (Jan 2018). <https://doi.org/10.1109/TPAMI.2017.2665623>
25. Tekin, B., Sinha, S.N., Fua, P.: Real-time seamless single shot 6d object pose prediction. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 292–301 (2018)
26. Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., Birchfield, S.: Deep object pose estimation for semantic robotic grasping of household objects. In: Conference on Robot Learning. pp. 306–316 (2018)
27. Vidal, J., Lin, C.Y., Martí, R.: 6d pose estimation using an improved method based on point pair features. In: 2018 4th International Conference on Control, Automation and Robotics (ICCAR). pp. 405–409. IEEE (2018)

28. Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., Savarese, S.: Densefusion: 6d object pose estimation by iterative dense fusion. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3343–3352 (2019)
29. Wu, J., Zhou, B., Russell, R., Kee, V., Wagner, S., Hebert, M., Torralba, A., Johnson, D.M.: Real-time object pose estimation with pose interpreter networks. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 6798–6805. IEEE (2018)
30. Xiang, Y., Schmidt, T., Narayanan, V., Fox, D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. arXiv preprint arXiv:1711.00199 (2017)
31. Xu, D., Anguelov, D., Jain, A.: Pointfusion: Deep sensor fusion for 3d bounding box estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 244–253 (2018)
32. Zhao, H., Shi, J., Qi, X., Wang, X., Jia, J.: Pyramid scene parsing network. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2881–2890 (2017)
33. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4490–4499 (2018)