Detecting Object Surface Keypoints from a Single RGB Image via Deep Learning Network for 6DoF Pose Estimation

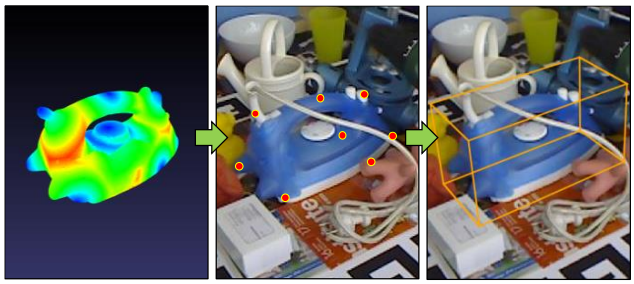
Lee Aing[1] and Wen-Nung Lie[1,2,3]
[1]Department of Electrical Engineering
E-mail: ainglee55@gmail.com
[2]Center for Innovative Research on Aging Society (CIRAS)
[3]Advanced Institute of Manufacturing with High-tech Innovations (AIM-HI)
National Chung Cheng University (CCU), Taiwan
Email: ieewnl@ccu.edu.tw

***Abstract*— Estimating the 6DoF object pose from a single RGB image is one of the challenging tasks in computer vision. Before the pose parameters can be defined by traditional PnP algorithm, 2D image projections of a set of 3D object keypoints have to be accurately detected. In this paper, we present techniques for defining 3D object keypoints and predicting their corresponding 2D counterparts via deep-learning network architectures. The main technique to designate object keypoints is firstly to employ k-means clustering for calculating the object surface weights and then select from all surface points the ones mostly distributive with larger surface weights to describe the object shape as possible. Moreover, Robust loss function is adopted in training the ResNet18 network for predicting image projection of object keypoints by focusing on small scale errors. Experimental results show that our proposed technique outperforms state-of-the-art approaches in ratio of correctness in both "2D projection" and "3D transformation" metrics.**

keywords: *6DoF object pose, deep learning, object shape.*

I. INTRODUCTION

The 6DoF (Degree of Freedom) object pose estimation is a process to identify the orientation and translation of the target object (with respect to a pre-built 3D model) in order to understand more about the 3D scenes. A precise pose estimation is always high-demanding for the variety of applications such as Virtual Reality (VR), Augmented Reality (AR), autonomous driving, Human Robot Interaction (HRI), etc. Things become more challenging when the estimation deals with only a single RGB image. Recently, one of the efficient approaches based on deep neural network to train and estimate eight corners of the 3D bounding box in a 2D image was proposed by [1] . Those estimated 2D corners were then used to align with the corresponding 3D corners by operating the PnP solver [2]. This proposed technique was simple but lacked the ability of identifying object shape information, because those bounding-box corners were not lying on the object surface. Besides designating keypoints [3] around the object surface to form a feature point set for describing the object shape is not sufficiently mature and still an open issue. The drawbacks of losing certain critical keypoints and learning naïve networks with emphasis on only large errors cause existing systems imperfect in terms of estimation accuracy.



(a) 3D keypoints (b) 2D keypoints (c) 6DoF pose

Fig. 1. Estimation of 6DoF object pose from 3D and 2D keypoints pairs. (a) Definitions of 3D object keypoints (blue regions mean the possible 3D keypoint after applying CPS), (b) estimated 2D keypoints which can be defined in (13), (c) 6DoF pose visualized as a 3D bounding box.

In this paper, we aim to fulfill those gaps by using the pixel-wise voting network invented by [3] as the base system and then enhance the network to achieve a higher performance. The first technique we adopt is to localize the critical object keypoints (Fig. 1(a)) by using a method called double direction keypoint sampling (DDKS). This method starts from grouping the object points as regions by utilizing the well-known K-Means clustering. From those cluster means (see Fig. 2(a)), a weight can be calculated for each object surface point in the cloud to seek for representative 3D keypoints. By using recorded camera parameters in the dataset (in this paper, LINEMOD is used), 2D projections of object keypoints can be produced, from which a set of unit-vector fields for all keypoints can be calculated and used as the ground truths for network training with cross entropy and Robust loss [4]. In testing, a procedure of keypoint voting is applied to restore 2D keypoints (Fig. 1(b)) from network outputs (unit-vector fields), and then the PnP algorithm [2] is used to recover the 6DoF pose (Fig. 1(c)) based on the paired correspondences of the 3D and 2D object keypoints. To summarize, main contributions of this research are:

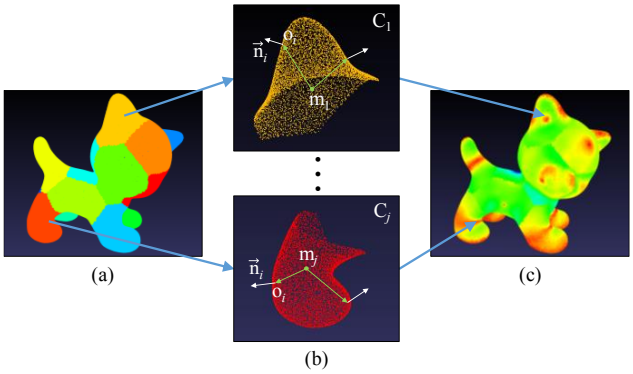


Fig. 2. Computing weights from DDKS. (a) All the object point clouds are grouped into regions by using K-Means. (b) Each clustering is applied by DDKS to define the weights. (c) After clustering weights are all computed, they are assembled back to become the 3D object weights.

- A direction-based technique (DDKS) to define 3D keypoints on the object surface which are capable of faithfully describing object shape for 6DoF pose estimation.
- A set of unit-vector fields with its gradients derived from 2D keypoints is used as ground truths for network training (not the 2D keypoints themselves).
- The combination of a pixel-wise voting network and the Ruber loss with the regularization term enables the network to upgrade the performance, especially in the 3D transformation metric.

II. Related Work

In this section, we will review some state of the art methods in defining 6DoF pose from a single RGB image. From the observation, we can basically divide those methods into two groups: pose-parameter based methods [5-8] and keypoint based methods [1, 3, 9, 10]. In this proposed method, the indirect keypoint based method is implemented to challenge with those state of the art methods, including some multi-stage training approaches (methods with refinement) whose performances are always difficult to defeat.

A. Pose Parameter Based Methods

One state of the art method proposed to estimate the pose parameters which were 3 vector translation and 4 quaternion rotation was presented by Billings *et al.* [8]. The main contribution of this approach was to store offline the 3D features in the network and then tried to repaint the occluded parts of the estimated object mask. From that complete object mask, the quaternion rotation was predicted combined with the prior translation to form the pose. The performance was competitive, but the system contained a lot of learning steps, which becomes complicated to train.

However, to enhance the performance, [5] [6] proposed techniques of refinement which were initialized by the initial pose from any existing techniques [7, 9] and outputted the pose offset. The differences between them were not only the object features using, but also model architecture. The refinement approach in [5] focuses on the optical flow and object mask to train the network while [6] put the effort on the object contour matching features with a new visual loss function. generally, they could obtain the better performance only if the refinement loops with more iterations. In conclusion, estimating the pose parameters might not be a good choice.

B. Keypoint Based Methods

Since the PnP algorithm was famous by solving 2D-3D correspondence object pose and consuming less time, [1] came up with the idea of regressing 2D projection of 3D object bounding box corners with DarkNet architecture from YOLO as the backbone. These proposed systems were feasible and fast enough, but low performance. This was because the estimation of those bounding box corners could not be robust to the occlusion problems and the corners were far away from the object, so that the training network gathered less correlation information between the object and the corners.

For the reason that the 2D projection points were the most important to estimate the object pose, Peng *et al.* [3] proposed a novel technique to indirectly estimate the 2D keypoints lying on the object surface. This seemed to be more plausible whereas the object surface keypoints contained more details than the corners, so that estimating can go more accurately. After restoring back the keypoints by using a voting technique, the covariances of those detected 2D keypoints were calculated and fed to an uncertainty PnP algorithm [2] for estimating the 6-DoF object pose. This indirect intermediate feature was great because they were easier to converge and post processing to infer the keypoints was not so complicated but precise.

III. Proposed Method

In this paper, we would like to estimate 2D projection keypoints, where their ground truth derived from 3D object keypoints based on a principle of double direction keypoint sampling (DDKS) and accordingly recorded camera parameters, and then the final step is to recover to estimate 6DoF pose. One of the indirect ways to estimate 2D keypoints is to estimate the unit vector field which is a pool of unit vector pointing from each pixel location to those keypoints. In training, the ResNet18 architecture is employed as the backbone, combined with skip connections of intermediate outputs, to get the desired up-sampled feature maps. In testing, the network outputs are reversely converted to get the estimated 2D keypoints by using a keypoint voting technique. Finally, the PnP algorithm is applied to estimate 6DoF object pose (see Fig. 3).

A. Double Direction Keypoint Sampling

Our definition of selecting 3D keypoints is not only to distribute the keypoints on the object surface as possible (e.g., like the farthest distance criterion in [3]), but also make them informative and representative for the object shape. In this technique, 3D curvature is considered as an important information used to seek for object keypoints. Firstly, the K-Means clustering technique is applied to assemble object point clouds or to cluster the object into regions (see Fig. 2(a)), in order to find a set of **K** cluster means (**K** has more number then

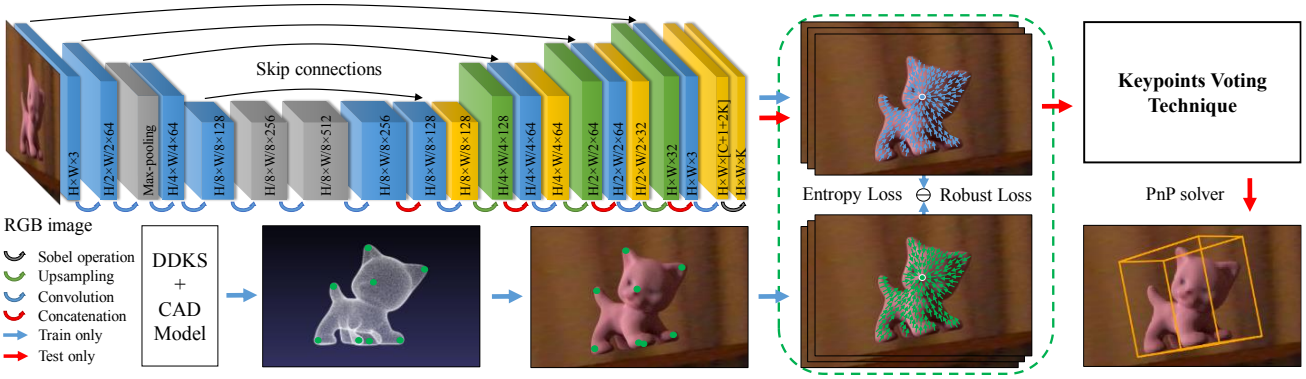


Fig. 3. Overview of the proposed system.. In the bottom row, the green points represent the selected keypoints on the object surface model and their corresponding 2D projections. The unit-vector fields are then calculated as the ground truths in network training. In the top row, ResNet18 is used as backbone network for predicting unit-vector fields from which the predicted 2D object keypoints in an image can be derived by passing through a keypoint voting technique. Those 2D keypoints are fed into the PnP solver for 6DoF pose estimation.

the object keypoints). Each object surface point is associated with a weight, which can be calculated from two vectors: the surface normal vector and the vector pointing from the corresponding cluster means to the object surface point itself. The combination of these two vectors can provide some information about the object shape, so that from that information we can choose the appropriate keypoints. The inner-product of these two vectors is calculated to obtain the included angle (0~π) between them, which is then normalized to [0, 1]. The weight of each point is expressed by:

$$w_i = 1 - \cos^{-1}\left(\frac{(\mathbf{o}_i - \mathbf{m}_j)^T}{\|\mathbf{o}_i - \mathbf{m}_j\|_2} \cdot \vec{\mathbf{n}}_i \right) \frac{1}{\pi}, \; i \in C_j, \tag{1}$$

where $\mathbf{o}_i$ and $\vec{\mathbf{n}}_i$ are the i-th object surface point and its corresponding normal vector, respectively, and $\mathbf{m}_j$ represents the means of j-th cluster C_j where $\mathbf{o}_i$ is located (see Fig.2 (b)).

To determine K keypoints, the selecting process is conducted in a subsequent manner. For each object point, its weight w_i (see Fig. 2(c)) is multiplied (element-wise multiplication) with a home distance d_i^{home} and temp distance d_i^{temp} as shown in (3). The home distance is defined as the distance between each object surface point and the object centroid. The object point with the largest weighted home distance will be selected as the keypoint. However, to make K keypoints more distributed when determining the next keypoint, the surrounding of the current keypoint is updated by a rule defined in (2). For each object surface point, a temp distance is calculated, which is defined as the distance of all points to the previous chosen keypoint. Fig.1(a) shows the keypoints are all defined from the high weighted surface (red surface) by avoiding the body which is covered by green color, and then they update themselves the surrounding to be low weighted surface (blue regions). This process is repeated until all K keypoints are identified.

$$D_i^{home} = \begin{cases} D_i^{home}, & \text{if } D_i^{home} \le D_i^{temp} \\ D_i^{temp}, & \text{otherwise} \end{cases}, \tag{2}$$

$$[D_i^{home}, D_i^{temp}] = [d_i^{home}, d_i^{temp}] \times w_i, \tag{3}$$

After projecting those 3D keypoints to obtain the corresponding 2D keypoints, a unit-vector field $\mathbf{v}_k$ (see green vector field in Fig. 3) for each 2D keypoint is calculated by using (4). All of the K unit-vector fields are used as the output ground truths for network training, which can be seen in Fig. 3 and will be described in part B in section III.

$$\mathbf{v}_k(\mathbf{p}_l) = \frac{\mathbf{s}_k - \mathbf{p}_l}{\|\mathbf{s}_k - \mathbf{p}_l\|_2}, \tag{4}$$

where $\mathbf{s}_k$ and $\mathbf{p}_l$ stand for the k-th 2D keypoint and the l-th pixel within the segmented object mask, respectively, and the outside object mask (no unit-vector field) is the background which is set to be zero value.

B. Deep Neural Network Design

The network backbone that we use to do the upsampling decoder is ResNet18. We further modify it by ignoring some down-samplings, so as not to lose much information in high level features. During forward pass, only four outputs with different down-sampling dimensions of the intermediate backbone outputs (the blue maps seen in Fig. 3) will be skipped and re-used to concatenate with the previous up-sampled feature maps (the green ones). The grey feature maps are convoluted with padding to have the same dimension but different depth. The concatenation is then passed through to the convolutional block (including 2D convolution, batch normalization, and activation function) before being up-sampled by using the bilinear interpolation to reproduce the same size as the corresponding skip connection feature maps. This simple process is continued until the desired output of size H×W×[C+1+2×K] is obtained, where H and W are image height and width, respectively, C and K are the numbers of object classes (1 is added because of the background) and keypoints, respectively, and the factor 2 comes from the fact that each unit vector is composed of horizontal and vertical components.

In network training process, cross entropy loss for measuring semantics in segmentation and Robust loss [4] for measuring correctness of unit-vector fields are employed. Robust loss contains two sub-loss functions, where one is the

Ruber loss proposed by Irie et al. [4] and the other is the Mean Absolute Error (MAE) loss. Ruber loss focuses on small scale errors rather than the large ones, meaning that higher priority will be given to optimize small scale errors which form the major part in the loss graph. Specifically, the combination of Ruber and MAE loss is accumulative in performance but less influence to each other. For training, Ruber loss is used to supervise the unit-vector fields first and then compute the corresponding gradients before passing through MAE loss that follows. The definition of Ruber loss and computation of gradients are expressed in (5) ~ (10) as below:

$$l_{Ruber}(e_{vector}) = \begin{cases} |e_{vector}|, & \text{if } |e_{vector}| \le c \\ \sqrt{2c|e_{vector}| - c^2}, & \text{otherwise} \end{cases}, \quad (5)$$

$$l_{MAE}(e_{gradient}) = |e_{gradient}|, \quad (6)$$

$$c = \max\{|e_{vector}|\} \times 50\%, \quad (7)$$

$$e_{vector} = \hat{v}_k(\mathbf{p}_l) - v_k(\mathbf{p}_l), \quad (8)$$

$$e_{gradient} = G_{\hat{\mathbf{v}}_k}(\mathbf{p}_l) - G_{\mathbf{v}_k}(\mathbf{p}_l), \quad (9)$$

$$G_{\mathbf{v}_k}(\mathbf{p}_l) = \sqrt{G^{x\,2}_{\mathbf{v}_k}(\mathbf{p}_l) + G^{y\,2}_{\mathbf{v}_k}(\mathbf{p}_l)}, \quad (10)$$

where $\hat{\mathbf{v}}_k(\mathbf{p}_l)$ and $\mathbf{v}_k(\mathbf{p}_l)$ are the estimated and the ground truth unit-vector (as defined in (4)), G^x and G^y are gradient operator along the x and y directions, respectively, e_{vector} and $e_{gradient}$ represent errors of unit-vectors and their gradient magnitudes, and c is a threshold defined to be 50% of the maximum absolute error in the mini-batch, which has a dimension of B×2K, where B is the batch size and K is the number of keypoints.

C. Derivation of 2D keypoints from estimated unit-vector fields

2D keypoints can be determined from the estimated unit-vector fields by voting and histogramming intersection points between any pairs of unit vectors in the pool. We follow the technique from [3] by changing parameter values and without computing the covariance. Firstly, the intersection points which are called keypoint candidates ($\mathbf{h}_{k,n}$, k = 1~K, n = 1~N) are determined, where N is the number of candidates for each keypoint. The pool of unit vectors which is also called the voter pool (v = 1~V) is random uniformly selected in the object mask, where V is the number of voters.

Next, the voting score which is defined in (11) [3] is calculated as the inner product [·] of the vectors pointing from pixels (the voter's locations) $\mathbf{p}_{k,v}$ to the keypoint candidate $\mathbf{h}_{k,n}$, and the estimated unit vectors of the voter pool $\hat{\mathbf{v}}_k(\mathbf{p}_{k,v})$ with the condition of being greater or equal to a pre-determined confidence threshold θ, where I[.] is a Boolean function that return 1 for true and 0 for false). The voting to each keypoint candidate from the voters is counted by doing the summation at the dimension v, and then the mask index of the best keypoint and candidate is determined by using the Boolean function I[.] before being element-wise-multiplied [∘] by $m_{k,n,v}$, and finally, the summation is performed to obtain the best mask voter for each keypoint $m_{k,v}$. This computation is expressed in (12).

$$m_{k,n,v} = \mathrm{I}\left[\frac{(\mathbf{h}_{k,n} - \mathbf{p}_{k,v})^T}{\|\mathbf{h}_{k,n} - \mathbf{p}_{k,v}\|_2} \cdot \hat{\mathbf{v}}_k(\mathbf{p}_{k,v}) \ge \theta\right], \quad (11)$$

$$m_{k,v} = \sum_n \left[m_{k,n,v} \circ \mathrm{I}\left[\sum_v m_{k,n,v} == \max_n\left(\sum_v m_{k,n,v}\right)\right]\right], \quad (12)$$

we apply this mask $m_{k,v}$ to define the best voters' locations $\mathbf{p}_{k,v} \circ m_{k,v}$ and best voters' unit vector fields $\hat{\mathbf{v}}_{k,v} \circ m_{k,v}$. In order to determine the estimated keyoints $\hat{\mathbf{s}}_k$, the linear equation of the orthogonal best voters' unit vectors and the vectors pointing from best voters' locations to keypoints can gets optimized by using least square fitting as seen in (13) [3]. To the end, the tolerant keypoints' locations are obtained. It is interesting to be noticed that this indirect (or voting-based) derivation of 2D keypoints from unit-vector fields is robust even in the presence of cropped object.

$$\hat{\mathbf{s}}_k = \frac{\left(\hat{\mathbf{v}}^{\perp}_{k,v} \cdot \mathbf{p}_{k,v}\right) \circ m_{k,v}}{\hat{\mathbf{v}}^{\perp}_{k,v} \circ m_{k,v}}, \quad (13)$$

IV. Experimental Results

A. Implementation Details

Our experiments were based on a platform of Core i9 CPU Intel processor with GeForce RTX 2080 Ti GPU in Ubuntu. The proposed method could be operated in average of 27 fps. Related to DDKS, 8 keypoints and an object centroid (K=9) are determined, and the value of c is fixed to be equal to 50% of the maximum absolute errors in the mini-batch. The number of epoch is set to be 100, so that some object models converging with higher epochs are not missed out and the training time is also reduced. To optimize the parameters, we use Adam as the optimizer with the batch size B = 8, and the learning rate starts from 0.001 and be adjusted by dividing in half per 20 epochs.

In the process of deriving keypoints from the estimated unit vector fields, the number of voters is set V = 100 points and the number of keypoint candidates is set to N = 200. These keypoint candidates are randomly re-paired when the probability is not satisfied or the iteration does not reach to the maximum. The threshold of probability of at least one inlier in the data samples is 0.99, and θ =0.999.

B. Evaluation Datasets and Metrics

To evaluate the proposed system, two LINEMOD datasets (normal and occlusion) are tested with two performance metrics such as 2D projection error (2D Projection), and average distance (ADD). The normal LINEMOD contains benchmark objects with clutter, texture-less, and poor lighting condition, while the occlusion LINEMOD focuses on heavy object occlusion. We follow the rule of evaluation, where

only 15% of the total images is used for training and the rest 85% is for testing which is applied for the normal LINEMOD, and for the occlusion LINEMOD, we use it for testing only. We also include synthetic images and data augmentation in the training with the different backgrounds obtained from PASCAL VOC [12].

Table 1. Performance (percentage of correctness) comparison in terms of "2D projection" and "ADD" metrics between our method and the baseline methods on normal LINEMOD dataset.

Method	2D Projection			ADD(-S)			
	[1]	[3]	Ours	[5]	[1]	[3]	Ours
ape	92.1	**99.2**	99.0	77.0	21.6	43.6	63.1
bvise	95.1	**99.8**	99.7	97.5	81.8	99.9	**99.9**
cam	93.2	99.2	**99.3**	**93.5**	36.6	86.9	90.7
can	97.4	**99.9**	99.8	96.5	68.8	95.5	**97.2**
cat	97.4	99.3	**99.7**	82.1	41.8	79.3	**83.9**
driller	79.4	96.9	**97.5**	95.0	63.5	96.4	**98.3**
duck	94.7	98.0	**99.0**	**77.7**	27.2	52.6	64.8
eggbox	90.3	99.3	**99.3**	97.1	69.6	99.2	**99.7**
glue	96.5	98.5	**99.1**	**99.4**	80.0	95.7	98.0
puncher	92.9	**100**	99.8	52.8	42.6	81.9	**89.9**
iron	82.9	99.2	**99.6**	98.3	75.0	98.9	**99.7**
lamp	76.9	98.3	**98.7**	97.5	71.1	99.3	**99.9**
phone	86.1	99.4	**99.6**	87.7	47.7	92.4	**95.5**
Average	90.4	99.0	**99.2**	88.6	55.9	86.3	**90.8**

Table 2. Performance (percentage of comparison in terms of "2D projection" and "ADD" metrics between our method and the baseline methods on occlusion LINEMOD dataset.

Method	2D Projection			ADD(-S)			
	[11]	[3]	Ours	[10]	[11]	[3]	Ours
ape	**69.6**	69.1	67.1	22.0	17.6	15.8	**25.2**
can	82.6	86.1	**90.3**	44.7	53.9	63.3	**71.1**
cat	65.1	**65.1**	64.9	**22.7**	3.3	16.7	21.0
driller	73.8	73.1	**76.9**	44.7	62.4	65.7	**72.7**
duck	61.4	61.4	**62.2**	15.0	19.2	25.2	**35.4**
eggbox	**13.1**	8.4	5.4	25.2	25.9	50.2	**52.0**
glue	54.9	55.4	**56.8**	32.4	39.6	**49.6**	46.5
puncher	66.4	69.8	**77.3**	**49.5**	21.3	39.7	46.7
Average	60.9	61.0	**62.6**	32.0	30.4	40.8	**46.3**

The "2D projection" measures the mean distance between the projections of all point clouds transformed by using the estimated and the ground truth poses. If the mean error is less than 5 pixels, the estimated object pose is considered correct. Moreover, we compute the ADD metric by transforming all point clouds with the estimated and the ground truth poses, and then calculating the mean distance between those two transformed point clouds in 3D space. If the average error is less than 10% of the ground truth object's diameter, the pose estimation is considered to be correct. For symmetrical objects i.e. "eggbox" and "glue", we use another metric called ADD-S [7].

C. Results and Analysis

The comparison in percentages of correctness in metrics of "2D Projection", and "ADD(-S)" between our proposed and the other state-of-the-art methods are shown in Table 1 for the normal LINEMOD dataset, respectively. From the experiments, we can see that our proposed method outperforms most of the other methods in those two metrics. The improvement is especially obvious in ADD metric owing to the adoption of Robust loss which optimizes the small scale errors which are the majority number in the total errors. This significant improvement behavior is quite similar when occlusion LINEMOD dataset is tested, which is shown in Table 2. This indicates that our proposed method is consistent to improve the performance in any conditions i.e. texture-less, occlusion, etc. In Table 2, we observe that one of the symmetrical objects which is "eggbox" has a very low performance. This is not because the detection accuracy is poor, in contrast, the precise pose which is inferred from its higher accuracy in 2D keypoint detection is reversed to the ground truth pose. It means that the symmetrical object in the training images and testing images are opposite. However, we can make the 2D projection performance of "eggbox" in Table 2 higher by just exchanging the detected keypoints from back to front, but the corresponding one in Table 1 will become low. In total, our proposed method cannot overcome the symmetrical objects.

V. CONCLUSIONS

We presented an improved technique to define critical 3D keypoints on object surface and use deep learning architecture to predict their 2D projections (i.e., 2D keypoints) in an image. Based on the corresponding pairs of 2D-3D keypoints, PnP solver is used to estimate 6DoF pose parameters. Our training technique relies on Robust loss to boost the performance up to another high level. A distinguishing performance about our proposed method is the improvement in 3D transformation (i.e., ADD) metric which is much higher, compared to the "2D projection" accuracy even in the occlusion situation.

ACKNOWLEDGMENT

This work was financially supported by the Center for Innovative Research on Aging Society (CIRAS), Advanced Institute of Manufacturing with High-tech Innovations (AIM-HI) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan.

REFERENCES

[1] B. Tekin, S. N. Sinha, and P. Fua, "Real-Time Seamless Single Shot 6D Object Pose Prediction," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 292-301.

[2] V. Lepetit, F. Moreno-Noguer, and P. Fua, "EPnP: An Accurate O(n) Solution to the PnP Problem," *International Journal of Computer Vision,* vol. 81, p. 155, 2008/07/19 2008.

[3] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "PVNet: Pixel-Wise Voting Network for 6DoF Pose Estimation," presented at the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019.

[4] G. Irie, T. Kawanishi, and K. Kashino, "Robust Learning for Deep Monocular Depth Estimation," in *2019 IEEE*

International Conference on Image Processing (ICIP), 2019, pp. 964-968.

[5] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep Iterative Matching for 6D Pose Estimation," *International Journal of Computer Vision,* 2019/11/05 2019.

[6] F. Manhardt, W. Kehl, N. Navab, and F. Tombari, "Deep Model-Based 6D Pose Refinement in RGB," in *Computer Vision – ECCV 2018*, Cham, 2018, pp. 833-849.

[7] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," *Robotics: Science and Systems (RSS),* 11/01 2018.

[8] G. Billings and M. Johnson-Roberson, "SilhoNet: An RGB Method for 6D Object Pose Estimation," *IEEE Robotics and Automation Letters,* vol. 4, pp. 3727-3734, 2019.

[9] Y. Hu, J. Hugonot, P. Fua, and M. Salzmann, "Segmentation-Driven 6D Object Pose Estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3380-3389.

[10] K. Park, T. Patten, and M. Vincze, "Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation," in *2019 IEEE International Conference on Computer Vision (ICCV)*, 2019.

[11] M. Oberweger, M. Rad, and V. Lepetit, "Making Deep Heatmaps Robust to Partial Occlusions for 3D Object Pose Estimation," in *Computer Vision – ECCV 2018*, Cham, 2018, pp. 125-141.

[12] M. Everingham, S. Eslami, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision,* vol. 111, 01/01 2014.