Object Detection and Pose Estimation using Rotatable Object Detector DRBox-v2 for Bin-Picking Robot

Eko Rudiawan Jamzuri, Agristia Riski Pinandita, Riska Analia, and Susanto
{ekorudiawan@polibatam.ac.id, pinandita69@gmail.com, riskaanalia@polibatam.ac.id}

Department of Electrical Engineering, Politeknik Negeri Batam, Batam, Indonesia

Abstract. This research aims to identify and estimate the object's pose to support bin-picking robot perception. In this research, we proposed the usage of the ArUco marker as a visual landmark of the detection area. Furthermore, the image of the detection area is processed by rotatable object detector DRBox-v2 to get the object's position and orientation in the camera frame. In the final process, the resulting DRBox-v2 position and orientation are transformed into a two-dimensional world coordinate as the final estimated pose. Based on the experimental result, the object detection yields an Average Precision (AP) of 0.54 while a threshold score of 0.5 is used. As the pose estimation result, the proposed method yields an average position error of 0.21 cm and a maximum position error of 0.28 cm. For the orientation error, the system achieves a maximum orientation error of about 1.23 degrees with an average orientation error of 0.58 degrees. This research contributes to the possibility of camera usage and end-to-end deep learning detector supporting bin-picking research.

Keywords: rotatable object detection, pose estimation, DRBox, deep learning, bin-picking

1 Introduction

A manipulator robot is used in the industry to do object manipulation tasks. The object manipulation process can be classified into two main tasks: picking and placing objects. In the picking process, the object will be picked randomly from a defined robot working area. The picking task success rate depends on the accuracy of the robot sensor in recognizing objects and estimating object pose. The higher error in pose estimation will cause the robot gripper to fail to grasp the objects. The process of recognizing objects, estimating the pose of objects, and picking up objects in a random environment is known as bin-picking.

The manipulator robot requires additional sensor integration to perform bin-picking tasks. The sensors commonly used for bin-picking applications are laser, 3D scanner, camera, and combination. The laser and 3D sensor detection results are pretty accurate for estimating the pose of an object. However, processing this sensor data requires high-cost computing because the sensor data is presented in a point cloud. In addition, the price of laser and 3D scanners is higher than the camera

ICAE 2022, October 05, Batam, Indonesia

DOI 10.4108/eai.5-10-2022.2326587

sensors. Currently, the use of camera sensors is the main alternative as a cheap sensor for bin-picking applications.

The camera sensor only produces image output, while the robot manipulator needs object class and estimated object pose as decision input. The vision system that can detect objects and estimate objects' pose needs to be emphasized to develop an excellent bin-picking arm robot. Cameras for bin-picking applications must be accompanied by an object recognition system capable of processing color images to produce outputs object position and orientation. The object's position can be represented as a translation vector concerning the reference coordinate system. At the same time, the object orientation can be described by a rotation matrix which states the object's rotation about a particular axis in the reference coordinate system. These outputs will be used as a reference for decision-making for the robot to carry out the task of taking objects.

Some researchers introduced many approaches to estimate the object pose using a camera. Based on the camera model, the approach can be generalized into two types: RGB or depth camera. [1], [2], and [3] presented the usage of RGB cameras for object pose estimation in bin-picking robots. The RGB camera usage needed a sequential process for estimating object pose. As presented in [1], the process is started by detecting the object's Region of Interest (ROI). [1] used YOLOv3, the improvement version of YOLOv1 [4] and YOLOv2 [5], as an object detector model to detect this ROI. Furthermore, the object's orientation is estimated using Fisher's linear discriminant from the resulting ROI coordinates. In contrast with [1], [3] used the same YOLO detector to detect the object directly. Then, the background removal process follows from this patch image object. Finally, [3] fed this post-processing image into a dual stream autoencoder to generate a 3D model of the object. An exciting approach is delivered by [2]. Instead of using a deep learning detector, [2] used a Histogram of Oriented Gradient (HOG) to detect objects in the image. Then, the object patch image is fed into Convolutional Neural Network to predict the object's position and orientation. Moreover, the CNN presented by [2] can predict the occluded and unoccluded objects.

Studies using the RGB-D camera approach are presented in [6], [7], [8], [9], and [10]. Generally, the approach is similar to estimating pose using an RGB camera. Deep learning detector models are still needed to segment the objects. The difference is in post-processing techniques, while depth image is used. For example, a study by [6] used YOLOv4 to detect and sort the object. From the YOLOv4 bounding box output, point cloud data is acquired from the depth image. The point cloud data is aligned with the 3D CAD model to estimate the object pose. [8] utilized similar techniques with [6]. However, the object is detected using MaskRCNN [11]. Instead of producing a bounding box, MaskRCNN yields a polyline representing the object's boundary. So, the extraction of point cloud data is more refined. Furthermore, in the pose estimation, [8] used Semantic Point Pair Feature (SPPF) techniques which are not needed a 3D CAD model reference. In contrast with [6] and [8], which used depth images for post-processing, [7] used depth images for pre-processing input images. The depth image is used for the background removal process. So, the image fed into the deep learning detector only consists of an object with a constant background. In this research, [7] proposed rotated RPN for object detection. The dual stream technique was introduced by [9], which used dual stream CNN to extract features from RGB and depth images in a single stage. However, point cloud processing is beside this stream to extract the 2D object regions. One technique is presented in [10], which is not needed depth image post-processing. The depth image is used directly

to measure the object's distance from the camera. So, the object's 3D position can be determined.

This study aims to detect and estimate object pose using a camera with deep learning approach. We proposed a deep learning rotatable object detector to detect the pre-trained objects and estimate their two-dimensional pose in the camera frame. Furthermore, we introduced the usage of the ArUco marker as the landmark of the Region of Interest (ROI). The recognized visual markers and objects are then combined to estimate object pose in the world coordinate.

The rest of this paper will be conducted as follows: Section II presents the material and method used in this research, which is the research dataset, object detector, and pose estimator, followed by the evaluation method of this research. Section III describes the overall experimental research results and will be closed with the conclusion and future work in Section IV.

2 Material and Method

This section will discuss the materials and procedures utilized during the research. This investigation commences with collecting image datasets of objects to be discovered. In addition, the rotatable object detection technique is employed to train the annotated data. The trained model will then be evaluated to determine the system performance and error rates.

2.1 Image Dataset

The first step in this research is collecting images from the research environment. We used the UR2 robot workspace area as the research environment shown in **Figure 1**. As seen in **Figure 1**, the robot is mounted at the top of the table. Besides, a camera is mounted pointing directly at the table surface. This camera will capture all objects on the table surface. We placed four ArUco markers on the table surface, which serve as landmarks to limit the camera detection area. Next, we used three text markers as experimental objects as our research testbed. We put these text markers inside a bin at the top of the table.

Figure 2 describes the procedure of collecting the image dataset in this research. The first step of dataset collection is placing text markers randomly inside the bin so that the position and orientation of text markers vary. Then, we triggered the camera to capture color images and recognized the ArUco marker with the OpenCV library. From the detected ArUco markers, we can obtain each marker's identity and coordinates. Next, these marker coordinates are used as references to transform and crop the detection area. The image transformation aims to change the image view to parallel with the camera, making it easier to scale from pixel units to distance units.

Equation 1 and 2 explained the perspective transformation, where the known parameters are the initial coordinates of point-i (x_i, y_i) and the desired coordinates of the point-i (x'_i, y'_i). Notation T is a 2D transformation matrix that serves to transform the point to the desired coordinate. In our case, four initial points are used in the transformation process. These points are taken from the ArUco marker center coordinates. As the desired coordinates, we defined fixed endpoints in the coordinates $(0,0)$, $(300,0)$, $(300,300)$, and $(0,300)$. Therefore, the transformation process will form a 300×300 pixels image representing the detection area. These endpoints are selected based on the object detection model's requirement, which needs fix-sized image input of 300×300 pixels.

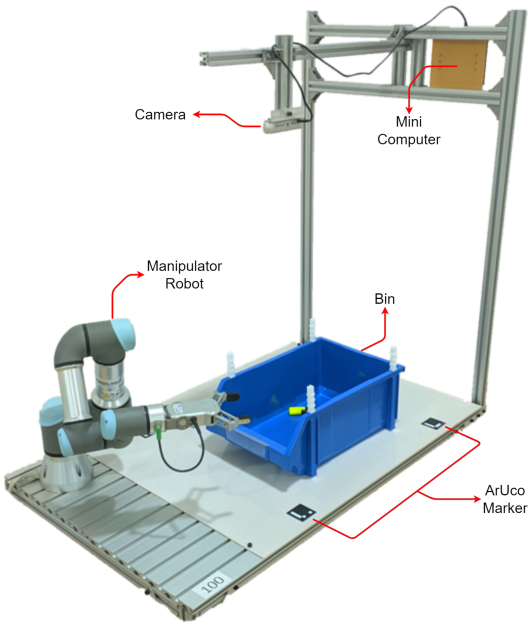


Fig. 1. Research environment.

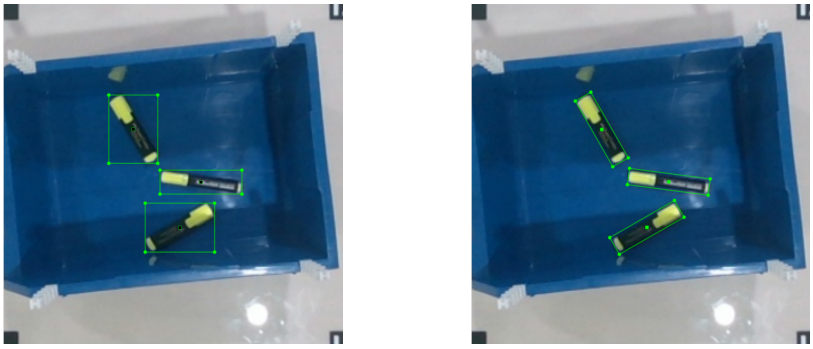
Fig. 2. Dataset collection procedure.

Finally, the formed image containing the detection area is saved into a file for the following data annotation process.

$$\begin{bmatrix} t_i x'_i \\ t_i y'_i \\ t_i \end{bmatrix} = T \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{1}$$

$$\text{dst}(i) = (x'_i, y'_i), \text{src}(i) = (x_i, y_i), i = 0, 1, 2, 3 \tag{2}$$

After the image collection process, we annotated these collected images using roLabelImg, an annotation tool capable of marking objects with a rotatable bounding box (RBox). In the annotation process, we marked all text markers with the same class, even though the dimension varies. As a result, the annotation process will yield an annotation file with file extension rbox. These files contain information about object center point location (x, y), object width (w), object height (h), object class (c), and object orientation (θ).



(a) **(b)**

Fig. 3. Annotated image dataset with (a) bounding box, (b) rotatable bounding box.

The total number of annotated images is 1100 images. We separated this image into two parts: 1. for the training and 2. for the testing/evaluation. The data we used for the training process was about 1000 samples, whereas the testing process was around 100 images. These 100 images will be tested and evaluated in the testing process to get performance metrics of the system.

2.2 Rotatable Object Detector DRBox-v2

The deep learning object detection methods generally yield bounding boxes as object identification results. In general, the bounding boxes have four parameters: the box center point (x, y), width (w), and height (h). However, this bounding box cannot be used to represent oriented objects perfectly, particularly if the object size and object orientation need to be estimated. **Figure 3** illustrates the limitation of bounding boxes used to mark oriented objects. The bounding box area is larger compared to the object area. Moreover, the orientation of the object is unknown. This limitation can be addressed using a rotatable bounding box described in **Figure 3**. As seen in **Figure 3**, the box area can represent the object area equally when using a rotatable bounding box. Moreover, the object's orientation can be determined by calculating the slope of the rectangle line. The rotatable bounding box in **Figure 3** is represented by the box center point (x, y), box width (w), box height (h), and additional orientation parameter (θ), which describes box orientation.

In this research, we used rotatable object detector DRBox-v2 [12] to detect the pre-trained objects. The selection is based on the capability of DRBox-v2 to yield a rotatable bounding box output so that the object orientation can be estimated. DRBox-v2 is Convolutional Neural Network (CNN) object detector initially proposed for detecting the target in the Synthetic-Aperture Radar (SAR) images. The architecture DRBox-v2 is similar to YOLO, which recognizes and localizes objects in a single stage. The difference is that YOLO used anchor boxes to sample positive and negative instances, while DRBox-v2 used prior RBox to sample positive and negative ones. The usage of prior RBox is to handle the limitation of anchor boxes, which only sample the objects with different aspect ratios without considering the object orientation.

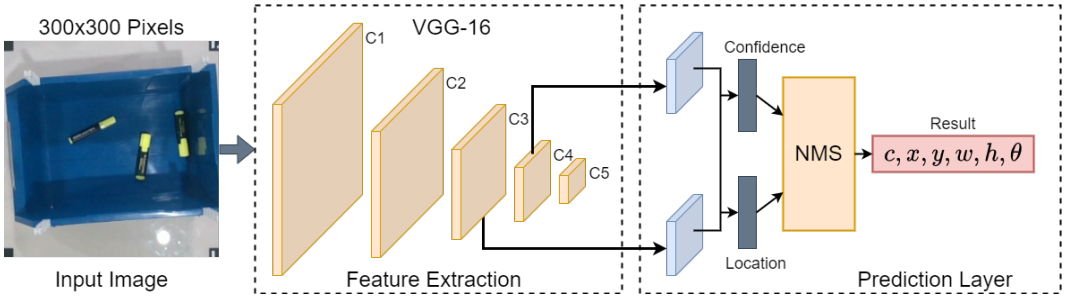


Fig. 4. DRBox-v2 Convolutional Neural Network architecture.

Figure 4 describes the detailed architecture of DRBox-v2. The input processed a fixed image size of 300×300 pixels through a Convolutional Neural Network (CNN), which consists of several layers. The function of this CNN is to extract features, while the last layer is to predict objects. DRBox-v2 used VGG16 architecture in the feature extractor layer. In the prediction layer, DRBox-v2 consists of K groups of channels, which described the number of previously defined RBox in each corner (prior RBox). In each prior RBox, the prediction layer calculated a confidence score, which indicated the target object's probability. In addition, the prediction layer has a 5-dimensional vector output that describes the offset value of the predicted RBox with the prior RBox. Next, the RBox is rotated according to the offset angles to predict the priors. Then it is predicted to determine the offset of the predicted RBox and RBox priors. Furthermore, this offset value must be decoded to produce an RBox that matches the predicted results.

The DRBox-v2 training procedure adopts the Single Shot Multibox Detector (SSD) [13]. In the training phase, the RBox in the dataset is grouped based on prior RBox. The grouping is based on the ArIoU value described in equation 3. ArIoU is a non-commutative function that calculates an oriented object's intersection area. In the training phase, grouping is done if the ArIoU value is $>$ 0.5. After that, the prior RBox is used as a positive sample to find the loss value L in equation 4, 5, and 6.

$$\mathrm{ArIoU}_{180}(A,B) = \frac{\mathrm{area}(\hat{A} \cap B)}{\mathrm{area}(\hat{A} \cup B)} \cdot |\cos(\theta_A - \theta_B)| \tag{3}$$

$$L(I,c,r,g) = \frac{1}{N_1 + N_2} L_{conf}(I,c) + \frac{1}{N_2} L_{loc}(I,K,G) \tag{4}$$

$$L_{loc} = \sum_{i \in \mathrm{Pos}} \sum_{j} \sum_{m \in \{x,y,l,w,\theta\}} I_{ij} \mathrm{smooth}_{L1}(k_j^m - g_j^m) \tag{5}$$

$$L_{conf} = -\sum_{i \in \mathrm{Pos}} (1 - c_i)^{\gamma} \log(c_i) - \sum_{j \in \mathrm{Hard\_Neg}} c_j^{\gamma} \log(1 - c_j) \tag{6}$$

Notation N is the number of matched rotatable bounding boxes, and the L_{conf} value is resulted from the softmax function between all selected positive and negative samples. Variable c is a two-

dimensional vector showing the confidence score's value. Finally, L_{loc} notation is a regression loss function used by DRBox-v2 to predict rotatable bounding box parameters.

2.3 Two-dimensional Pose Estimation

While the robot executes the picking task, the object pose represented in the world coordinate has to be known. However, the DRBox-v2 represents object position in the camera frame coordinate with pixel units. Therefore, object position conversion from camera frame to world coordinate is needed. As described in **Figure 1**, our environment and camera position are fixed. Moreover, we put visual landmarks on the table surface. Where the distance between landmarks is known, both in centimeter and pixel, so the scaling factor in x-coordinate (s_x) and y-coordinate (s_y) can be determined. Then from these scaling factors, the two-dimensional object's position in the world coordinate (x_w, y_w) can be defined by equation 7. Where x_c and y_c represent the two-dimensional object's position in the camera frame.

$$\begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} \tag{7}$$

2.4 Evaluation Method

We used the Average Precision (AP) metric to evaluate the results of our study. AP value describes the detector's average accuracy under varying recall rates [14]. The higher the AP value, the better detector performance. In order to derive the AP value, first, the model precision rate (P_d) is calculated using equation 8, where N_{td} describes the total number of true positive predictions and N_d defines the total number of all predicted objects. Furthermore, the recall value (R_d) is calculated using equation 9 by dividing N_{td} with the total number of actual objects (N_r). Then from the resulted P_d and R_d, the AP value can be determined using the point interpolation method introduced in [15].

$$P_d = \frac{N_{td}}{N_d} \tag{8}$$

$$P_d = \frac{N_{td}}{N_d} \tag{9}$$

$$\theta_{err} = \text{asin}(\sin(|\hat{\theta} - \theta|)) \tag{10}$$

Instead of only measuring object detection metrics, we also analyzed the performance of 2D pose estimation. The evaluation can be done by comparing actual and predicted object poses. We measure the proposed system performance using position and orientation errors. The position error can indicate the displacement of the actual object with the predicted object position in the world coordinates. We used the Euclidean distance method to represent this error. The lower the Euclidean distance value, the better system performance. Moreover, the orientation error (θ_{err}) can be determined using equation 10, which represents the norm of absolute orientation error of predicted object orientation ($\hat{\theta}$) and actual object orientation (θ).

3 Result and Discussion

In this chapter, we will describe the results of this research. First, the results of object detection and pose estimation will be presented, followed by the value of the evaluation metrics. In addition, we also measure the required computational speed of the proposed system.

3.1 Training and Evaluation Result

In the training process, we used a computer with an NVIDIA Graphical Processing Unit (GPU) GTX 1080. Moreover, we change the default configuration setting of DRBox-v2 to match the dataset's object dimension. Due to the camera position being fixed, the captured object will result in a constant dimension. So, we set the prior RBox height and width interval with fixed values, 15 and 90 pixels. These values are determined after analyzing the statistically annotated RBoxes dimension in the dataset. For the prior RBox orientation, we follow the default value of the DRBox-v2 orientation interval of 30 degrees. This configuration will result in a total of 84828 prior RBoxes during training. The pre-processing time to extract these prior RBoxes is around 23 minutes. Furthermore, we use batch size = 16 in the training process to maximize GPU memory usage. Moreover, the training process takes a maximum of 50000 epochs.

We store the loss function historical value to evaluate the training results of the DRBox-v2 model. **Figure 5** shows the training performance while the model trained until 20000 epochs. It can be seen from the curve that the value of the loss function is decreasing. This graph shows that the learning process successfully minimizes the loss function. For example, it can be seen on the curve that the loss value at the beginning of the training was in the upper of 5 for total loss, location loss, and confidence loss. After the 2500 epoch, all loss values decreased drastically to below 1. In our historical data, the total loss value is 0.004, the location loss value is 0.0039, and the confidence loss value is 0.0001 in the last epoch.

After evaluating the training results, we validated the trained model to the test dataset. We varied the confidence score threshold from 0-1 with an interval of 0.1 to study the effect of the threshold value on the AP score. **Table 1** illustrates the relationship between the confidence score threshold and the AP score. The resulting AP score is above 0.5 if the confidence score threshold is below or equal to 0.7. The AP score will decrease significantly if the confidence score is set above that value. Moreover, the lowest confidence score threshold will result in a higher AP score. We get the highest AP score of 0.69 and the lowest AP score of 0.01.

Table 1: Resulted average precision over different score threshold.

AP 0.1	**AP 0.2**	**AP 0.3**	**AP 0.4**	**AP 0.5**	**AP 0.6**	**AP 0.7**	**AP 0.8**	**AP 0.9**
0.69	0.54	0.54	0.54	0.54	0.52	0.51	0.28	0.01

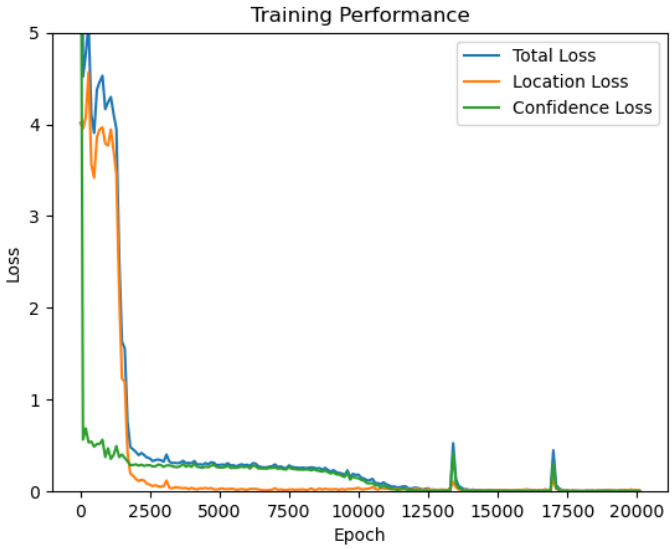
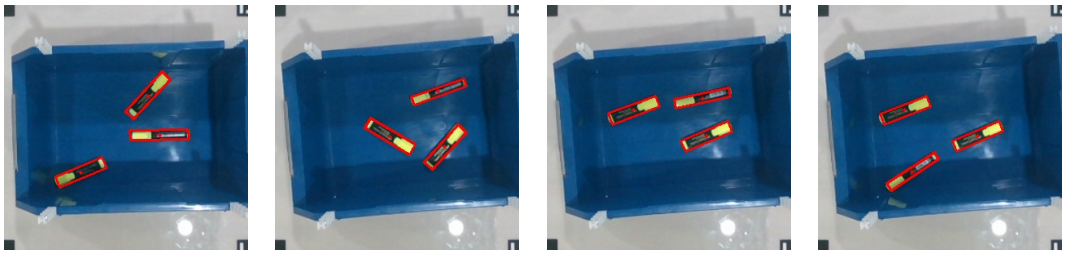


Fig. 5. DRBox-v2 training performance.

3.2 Object Detection and Pose Estimation Result

Figure 6 describes the object detection results from the DRBox-v2 model. In this experiment, we set the confidence score threshold to 0.1. The result of object detection is marked with a red RBox. In the center of the RBox, we put a red dot indicating the objects' center point. As seen in **Figure 6**, all text markers have been marked with RBox, indicating that the detector identified the object successfully. In addition, the resulting RBox orientation exactly matches the object's orientation.



(a) **(b)** **(c)** **(d)**

Fig. 6. Detection result on the test dataset.

After evaluating the object detection, we analyze the estimated pose estimation of the system. The estimated pose from the proposed system is presented in **Table 2**. The performance of the pose estimator can be seen in the column position error and orientation error. **Table 2** shows that the proposed method yields an average position error of 0.21 cm and a maximum error of 0.28 cm. Furthermore, as orientation error, the proposed system achieves an average orientation error of 0.58 degrees, with the highest error of 1.23 degrees. This result indicates that the proposed system can estimate object pose accurately.

Table 2: Estimated pose from DRBox-v2 prediction output.

Image	**Estimated Pose**			**Actual Pose**			**Pos. Err.**	**Ori. Err.**
	x **(cm)**	y **(cm)**	θ **(deg)**	x **(cm)**	y **(cm)**	θ **(deg)**	**(cm)**	**(deg)**
	38.40	32.00	-0.81	38.60	31.80	177.71	0.28	0.55
Fig. 6 (a)	35.40	22.20	42.73	35.60	22.00	42.40	0.28	0.33
	18.80	41.00	157.44	18.60	41.00	158.23	0.20	0.79
	40.20	21.20	160.60	40.40	21.20	159.37	0.20	1.23
Fig. 6 (b)	41.60	34.80	132.81	41.60	34.80	41.83	0.00	0.13
	28.40	32.40	30.29	28.20	32.20	32.66	0.28	0.77
	38.40	22.80	-11.87	38.60	22.60	166.25	0.28	0.95
Fig. 6 (c)	21.40	25.60	159.30	21.20	25.80	158.80	0.28	0.50
	39.20	32.00	157.16	39.20	32.00	157.08	0.00	0.08
	23.20	40.80	146.47	23.00	41.00	145.62	0.28	0.85
Fig. 6 (d)	21.40	25.80	158.88	21.20	25.80	158.80	0.20	0.08
	39.20	32.00	156.89	39.20	32.20	157.65	0.20	0.76
			Average				**0.21**	**0.58**

3.3 Computation Time

We also evaluated the computation time besides experimenting with system accuracy. In this experiment, we compare two processing devices' performance; 1. a computer desktop that is integrated with GPU, and 2. a mini-computer that only has CPU as the processing device. The specification detail of these computing devices is described in **Table 3**. The purpose of comparing these computing devices is to determine if the proposed system could be implemented on the low computing device.

Table 3 describes the resulting inference time for processing an image until getting the object pose. While experimenting on the desktop computer equipped with GPU, we got an average computation of around 38.51 ms for one image. Meanwhile, if we experimented with a mini-computer, processing time would be increased by about 16 times. The mini-computer needs 631.22 ms to process one image until getting the result. This result concludes that there is a possibility to implement the proposed system on CPU devices. However, the computation speed needs to be reduced for a real-time system. So, it can be concluded that GPU is needed to get real-time performance.

Table 3: Computation time over different computing devices.

Computing Devices	Computation Time (ms)
CPU : Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz **CPU Core** : 8 cores, 2 thread per core **RAM** : 16 GBytes **GPU** : NVIDIA GTX1080	38.52
CPU : Intel(R) Core(TM) i3-5010U CPU @ 2.10GHz **CPU Core** : 4 core, 2 thread per core **RAM** : 4 GBytes **GPU** : None	631.22

4 Conclusion and Future Work

This paper proposed an object detection and poses estimation system using rotatable object detector DRBox-v2 and ArUco visual marker. Based on the experimental result, the proposed system accurately detects the object with an AP of 0.54 while the confidence score threshold is set to 0.5. Moreover, the proposed system yields a low error on the estimated pose. Based on the actual and estimated pose measurement, the average position error is about 0.21 cm, and the average orientation error is about 0.58 degrees. There is an opportunity to improve the computational speed and detection accuracy in future works, mainly when the system is run in real-time on low-speed CPUs. Currently, the system yielded 16 times slower processing time while implemented on the low-speed CPUs. Additionally, this research represented the object pose with a two-dimensional pose. In a complex environment with barrier objects available, the three-dimensional object pose needs to be represented to make the robot possible to pick the object from a possible direction.

Acknowledgment

This research is part of the Research and Community Service for Vocational Higher Education research grant and the Final Project at the Robotics Engineering Study Program, Department of Electrical Engineering, Politeknik Negeri Batam. We thank Direktorat Jenderal Pendidikan Vokasi, Politeknik Negeri Batam, and Barelang Robotics and Artificial Lab (BRAIL) for providing the equipment and facilities for this research.

References

[1] Mou F, Ren H, Wang B, Wu D. Pose estimation and robotic insertion tasks based on YOLO and layout features. Engineering Applications of Artificial Intelligence. 2022 sep;114(January):105164.

[2] Kozák V, Sushkov R, Kulich M, Přeučil L. Data-Driven Object Pose Estimation in a Practical Bin-Picking Application. Sensors. 2021 sep;21(18):6093.

[3] Lee S, Lee Y. Real-Time Industrial Bin-Picking with a Hybrid Deep Learning-Engineering Approach. In: 2020 IEEE International Conference on Big Data and Smart Computing (BigComp). IEEE; 2020. p. 584-8.

[4] Redmon J, Divvala S, Girshick R, Farhadi A. You Only Look Once: Unified, Real-Time Object Detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE; 2016. p. 779-88.

[5] Redmon J, Farhadi A. YOLO9000: Better, Faster, Stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE; 2017. p. 6517-25.

[6] Wong CC, Tsai CY, Chen RJ, Chien SY, Yang YH, Wong SW, et al. Generic Development of Bin Pick-and-Place System Based on Robot Operating System. IEEE Access. 2022 jun;10:65257-70.

[7] Yao F, Wang S, Li R, Chen L, Gao F, Dong J. An accurate box localization method based on rotated-RPN with weighted edge attention for bin picking. Neurocomputing. 2022 apr;482(xxxx):264-77.

[8] Zhuang C, Wang Z, Zhao H, Ding H. Semantic part segmentation method based 3D object pose estimation with RGB-D images for bin-picking. Robotics and Computer-Integrated Manufacturing. 2021 apr;68:102086.

[9] Soltan S, Oleinikov A, Demirci MF, Shintemirov A. Deep Learning-Based Object Classification and Position Estimation Pipeline for Potential Use in Robotized Pick-and-Place Operations. Robotics. 2020 aug;9(3):63.

[10] Zhang H, Tan J, Zhao C, Liang Z, Liu L, Zhong H, et al. A fast detection and grasping method for mobile manipulator based on improved faster R-CNN. Industrial Robot: the international journal of robotics research and application. 2020 jan;47(2):167-75.

[11] He K, Gkioxari G, Dollar P, Girshick R. Mask R-CNN. In: 2017 IEEE International Conference on Computer Vision (ICCV). vol. 2017-Octob. IEEE; 2017. p. 2980-8.

[12] An Q, Pan Z, Liu L, You H. DRBox-v2: An Improved Detector With Rotatable Boxes for Target Detection in SAR Images. IEEE Transactions on Geoscience and Remote Sensing. 2019 nov;57(11):8333-49.

[13] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, et al. SSD: Single Shot MultiBox Detector. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). vol. 9905 LNCS. Springer Verlag; 2016. p. 21-37.

[14] Padilla R, Netto SL, da Silva EAB. A Survey on Performance Metrics for Object-Detection Algorithms. In: 2020 International Conference on Systems, Signals and Image Processing (IWSSIP). vol. 2020-July. IEEE; 2020. p. 237-42.

[15] Everingham M, Van Gool L, Williams CKI, Winn J, Zisserman A. The Pascal Visual Object Classes (VOC) Challenge. International Journal of Computer Vision. 2010 jun;88(2):303-38.