Real-Time Instance Segmentation and Point Cloud Extraction for Japanese Food

Proceedings of the SICE Annual Conference 2020
September 23-26, 2020, Chiang Mai, Thailand

Real-Time Instance Segmentation and Point Cloud Extraction for Japanese Food

Suthiwat Yarnchalothorn[1], Nattapol Damrongplasit[1], Sakmongkon Chumkamon[2], and Eiji Hayashi[2]

[1]Department of Mechanical Engineering, Faculty of Engineering,
Chulalongkorn University 254 Phayathai Road, Pathumwan, Bangkok 10330 Thailand
(E-mail: joestw1996@gmail.com)
[2]Mechanical Information Science and Technology, Kyushu Institute of Technology,
680-4 Kawazu, Iizuka-shi, Fukuoka, 820-8502, Japan

Abstract:
Innovation in technology is playing an important role in the development of food industry, as is evidenced by the growing number of food review and food delivery applications. Similarly, it is expected that the process of producing and packaging food itself will become increasingly automated through the use of a robotic system. The shift towards food automation would help ensure quality control of food products and improve production line efficiency, leading to reduced cost and higher profit margin for restaurants and factories. One key enabler for such automated system is the ability to detect and classify food object with great accuracy and speed. In this study, we explore real-time food object segmentation using stereo depth sensing camera mounted on a robotic arm system. Instance segmentation on Japanese food dataset is used to classify food objects at a pixel-level using Cascade Mask R-CNN deep learning model. Additionally, depth information from the sensor is extracted to generate a 3D point cloud of the food object and its surroundings. When combined with the segmented 2D RGB image, a segmented 3D point cloud of the food object can be constructed, which will help facilitate food automation operation such as precision grasping of food object with numerous shapes and sizes.

Keywords: Instance Segmentation, Cascade R-CNN, 3D Point Cloud, Food Automation

1. INTRODUCTION

Food industry is an important part of the economy for many countries. It represents a large portion of the country's GDP, and it is made up of various business sectors that provide food supply to the population, including agriculture, manufacturing, food processing, and food services, just to name a few. By incorporating innovative technologies to these existing sectors, the overall efficiency can be improved, while the operating cost can be reduced. One such example is the use of automation in the food production process. Automating food production can help to improve consistency in the appearance and quality of the food being produced, as well as to minimize unnecessary waste by using minimum amount of ingredients. Although the benefit of automation in food production is clear, there has not been a wide-scale adoption of such technology in the food industry, as one might observe in other industries like car or industrial manufacturing. One reason has to do with the ability to manipulate food objects. Unlike other industries where the components are usually uniform in size and weight, the ingredients that go into food production are often of diverse size, shape, weight, and texture, making it difficult to develop a line automation using traditional methods [1].

Computer vision may help solve these challenging problems, however. The technology mimics how human sees and perceives things. One important aspect of computer vision is image recognition, which involves different processes such as object detection, classification, and segmentation. A particular technique, called instance segmentation, has been used to detect distinct objects in an image in real-time and the classification is done at a pixel-level using deep learning model. Previous studies have shown this approach to be effective and reliable in detecting food objects [2, 4, 5].

In addition to the output RGB data from a camera sensor, depth camera technology can also provide 3D depth information which can be advantageous for analyzing 3D objects in a surrounding environment. By combining the segmented RGB data and 3D depth information together, automatic detection and classification of food object can be done more accurately, allowing for better estimation of its shape, weight, size and, even calories.

The main contribution of the paper is to explore instance segmentation and point cloud extraction for food object using data from a single stereo depth camera. In this study, we use the data from a single stereo depth camera mounted on a robot end-effector which is used for performing tasks involving food automation. The outputs from the camera are RGB color data and depth information. We perform instance segmentation on the RGB data by using Cascade Mask R-CNN which is a powerful deep learning architecture based on mmdetection [3] benchmark result with the box AP and mask AP of 41.2% and 35.9% on COCO 2017 test-dev dataset [4], respectively. The Cascade Mask R-CNN method is a combination of Cascade R-CNN [4] and Mask R-CNN [5], which allows classification of RGB data at a pixel level. Then, we combine the processed data with depth information to achieve 3D object segmentation. All of the output information from our proposed process is published on a robot system network that can later be used in path planning or grasping posture estimation. Moreover, the resulting output can be generalized to other systems that are equipped with a depth sensing camera, not just limited to a robotic system.

2. EXPERIMENTS

2.1 Robot System

The robot system used in this experiment is a 7-Axis articulated arm robot system from Yaskawa Electric Corporation, Japan, with a model name Motoman SIA5F [6]. It supports many applications such as assembling, machine tending, material handling, part transferring, and picking-and-placing with a payload of up to 5.0 kg. It has a horizontal reach of 559 mm and a vertical reach of 1007 mm. The controller model is FS100. The robotic arm is connected to and controlled by several computers used for performing different tasks such as path planning and gripper controlling. The entire robot system is operated and communicated on Robot Operating System (ROS) network.

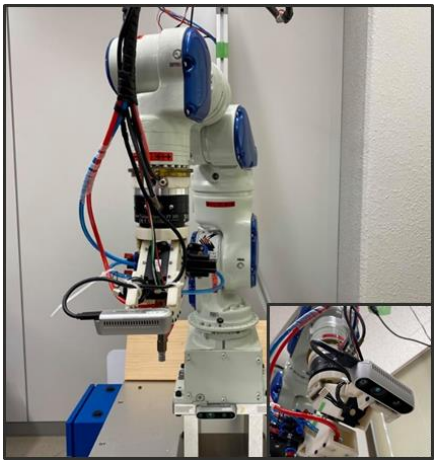

Fig.1 The robot system with a stereo depth camera mounted at the end-effector.

Several depth cameras are set up in the system to perform workspace calibration and computer vision. One of the cameras fixed near the end-effector is a stereo depth camera. It is used for computer vision to enhance object manipulation task like pick-and-place with a robotic arm. The model of this depth camera used is Intel RealSense D435i [7]. It is connected via a USB to a desktop computer running high performance GPU, Nvidia GeForce RTX 2080Ti, which is suited for performing high computation tasks like deep learning [8].

The data flow for the image processing of our system is shown in Fig.2. Computer A directly receives the raw data from the depth camera which includes RGB and depth data. The RGB data are used for instance segmentation and then combined with the depth data to create 3D point cloud information. The processed data are sent to other computers in the network to be used in other processes like path planning and grasp posture optimization.

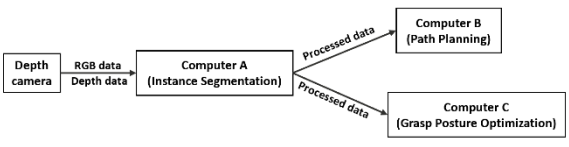

Fig.2 Data flow example of image processing part in robot system.

2.2 Dataset

The deep learning is conducted on our curated image dataset annotated in the same style as COCO dataset via COCO-Annotator [9,10]. The dataset consists of 807 images, of which 765 images are annotated, while the rest are images of the laboratory environment. There are 13 categories in the dataset: Japanese lunch box (Bento Box), bologna sausage, potato chip, Japanese fried chicken (Karaage), Japanese rice ball (Onigiri), and 8 different kinds of sushi which are shrimp (Ebi), squid (Ika), red caviar (Ikura), salmon, Japanese omelette (Tamago), tuna, eel (Unagi), and sea urchin roe (Uni). The detail of the dataset is summarized in Table 1. Examples of annotated images of the food objects belonging to the different categories are shown in Fig.3.

Table 1: Dataset detail.

Category	Images		Annotations	
	Train	Val	Train	Val
Bento box	142	17	142	17
Bologna	27	3	195	23
Potato chip	55	9	82	11
Ebi nigiri	105	11	139	17
Ika nigiri	101	10	138	13
Ikura nigiri	49	6	84	9
Karaage	104	16	474	59
Onigiri	96	10	325	27
Salmon nigiri	103	13	197	21
Tamago nigiri	98	10	159	19
Tuna nigiri	53	4	117	7
Unagi nigiri	49	5	118	11
Uni nigiri	49	6	104	12
Surroundings	38	4	-	-

Fig.3 Examples of annotated food images.

2.3 Deep Learning Implementation

In this experiment, we use mmdetection [3] as a framework for performing deep learning. The architecture used for training is Cascade Mask R-CNN, implemented with PyTorch [11] and mmdetection. The backbone is ResNet-50. The detectors are trained on a single GeForce RTX 2080Ti (three images per GPU) for 1000 epochs with an initial learning rate of 0.02. The trained model can also be applied for testing in the same framework.

2.4 Point Cloud Extraction via Depth Topic

The depth camera streams data which consist of RGB and depth frames that can be used to generate point cloud ROS message. We use *realsense2_camera* package to access data from the camera. The package's node, called */camera/realsense2_camera,* publishes raw camera data which are RGB, depth, and camera information to different ROS topics. The */pointcloud_masking* node subscribes to these topics and accepts the data as inputs which will then be transformed into point cloud information of food objects. The subscriptions are done via callback functions. The RGB data are used to perform inference with the trained model, providing detection and segmentation information (mask) of the food objects being detected. These masks are binary masks in an array structure with the same dimension as the original RGB image, but the values in the array are stored in Boolean format as true or false instead. The true value indicates that the pixel is in a segmented object, and vice versa. This binary mask is then applied to the depth data from the subscribed depth topic. The depth data is an array consisting of depth value for each pixel and it has the same dimension as the binary mask array. This processing step is demonstrated with a 6x8 size image as shown in Fig.4, where Fig.4a represents a mask array, Fig.4b represents a depth frame, and Fig.4c represents the resulting depth frame after applying the mask array.

F	F	F	F	F	F	F	F
F	F	F	T	T	F	F	F
F	F	T	T	T	T	F	F
F	F	T	F	T	T	F	F
F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F

(a) 6x8 Mask array

Z11	Z12	Z13	Z14	Z15	Z16	Z17	Z18
Z21	Z22	Z23	Z24	Z25	Z26	Z27	Z28
Z31	Z32	Z33	Z34	Z35	Z36	Z37	Z38
Z41	Z42	Z43	Z44	Z45	Z46	Z47	Z48
Z51	Z52	Z53	Z54	Z55	Z56	Z57	Z58
Z61	Z62	Z63	Z64	Z65	Z66	Z67	Z68

(b) 6x8 Depth frame

0	0	0	0	0	0	0	0
0	0	0	Z24	Z25	0	0	0
0	0	Z33	Z34	Z35	Z36	0	0
0	0	Z43	0	Z45	Z46	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

(c) Depth frame after applying the mask

Fig.4 Process of applying mask to a depth frame

The depth values in the array are in millimeters and its actual dimensions are H x W x 1, where H and W represent the height and width of the image, respectively. After applying the mask to the depth frame, the depth value outside the mask area will be set to zero. At this point, we have retained only the depth information of the desired area. The resulting depth frame is then transformed from a pixel coordinate into a metric coordinate in order to create 3D point cloud. To accomplish this, we need the camera intrinsic parameters which are used to map camera coordinates into image plane (world points). The camera intrinsic parameters are obtained from */camera/color/camera_info* ROS topic via a callback function. The metric coordinate can be calculated according to Eq. (1).

$$z = depth/1000$$
$$x = z(u - K_{13})/K_{11} \quad (1)$$
$$y = z(v - K_{22})/K_{21}$$

, where x, y, z are the metric coordinates of a point, u and v are the horizontal and vertical pixel coordinates of a point, and K is the camera intrinsic matrix, respectively.

Once we have obtained a set of 3D points in metric coordinates, we are able to create *sensor_msgs/PointCloud2* to publish the information in the ROS system. In this case, the point cloud information of each category is published on different topics. For examples, */pointcloud_1* is a point cloud topic for category one, and */pointcloud_2* is a point cloud topic for category two. Furthermore, we can colorize the point cloud for each category to enhance visualization by adding a floating RGB value to each point in the set of 3D points, so that each point will now contain x, y, z, and an RGB value. The ROS diagram for this node is shown in Fig.5.

2.5 Centroid of Point Cloud

In the previous section, we discussed how individual point clouds are extracted from a depth camera data. Using the extracted point cloud data, we can calculate the centroid position of a point cloud by simply taking the average values of each coordinates x, y, and z values belonging to the segmented object. Then, we publish the centroid position of the segmented object in a ROS topic using */visualization_msgs/Marker* ROS message. This information is especially useful for food automation involving pick-and-place operations.

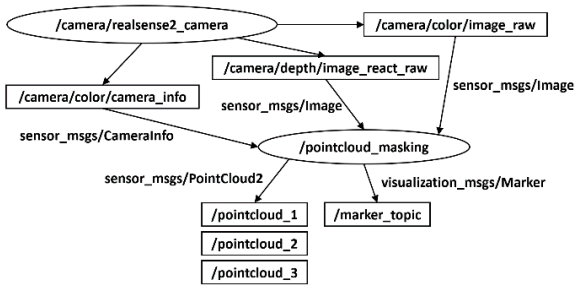

Fig.5 First method for point cloud extraction in ROS via Depth Topic

2.6 Point Cloud Extraction via Point Cloud Topic

In the previous section, we explained one method to obtain food object's point cloud information and publish that information in a ROS environment. There is also an alternative method that can be used to extract point cloud as illlustrated in Fig.6.

This second method differs from the first method in the way that it subscribes to the point cloud topic instead of the depth topic. One advantage of using this second method is that it simplifies the calculation in the */pointcloud_masking* node. If a point cloud topic provides the depth registered point cloud message, we can easily obtain the point cloud of the desired objects by

applying the mask array to the point cloud array directly. The depth registered point cloud is an array with a dimension of H x W x 4. The number 4 implies each point has 4 attributes, which are x, y, z, and RGB values of a point. With this kind of point cloud information, we can apply a mask to it in a similar manner as previously shown in Fig.4. However, if a point cloud is not depth registered, we will need to perform additional calculation. For example, after the inference step, we find all the possible contours (in a closed form) in the mask image and pick out the largest contour. Next, we generate a polygon of this contour, consisting of a set of vertices of the contour boundary. Then, we filter the desired points by determining only the points in the points array that reside in the polygon. Each point in the point cloud contains an RGB color from the camera's RGB data. Hence, the object in the point cloud format can be displayed with its true color instead of a mono color as is the case with the first point cloud extraction method.

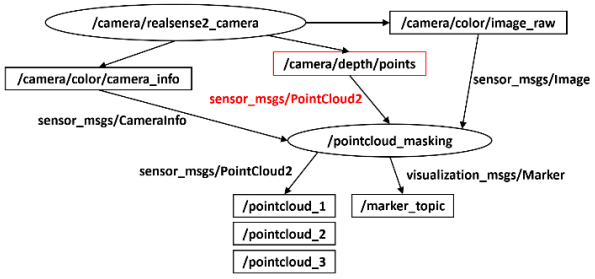

Fig.6 Second method for point cloud extraction in ROS via Point Cloud topic.

3. RESULTS AND DISCUSSION

3.1 Instance Segmentation Model

The trained instance segmentation is tested with a set of 87 images with the score threshold of 0.8. The result is evaluated using mean average precision metric (mAP). This model achieves bbox AP and mask AP of 66.9% and 68.7%, respectively. This indicates that the deep learning model is able to successfully detect and classify different food objects in the test images. An example of a segmentation result of an RGB image consisting of different food objects is shown in Fig.7.

Fig.7a RGB images of food objects.

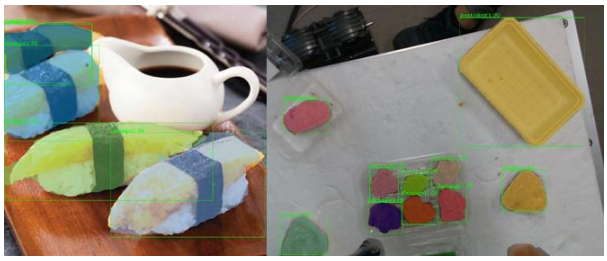

Fig.7b Segmentation results from the RGB images.

3.2 Point Cloud Extraction

Fig. 8 (Left) shows an RGB frame of the depth camera and Fig. 8 (Right) shows the corresponding extracted point cloud of that frame with the depth cloud of the surrounding environment. Using *rviz* for visualization, the point cloud messages of each object are published in different ROS topics with different colors and they can be visualized simultaneously. In this example image, there are 4 objects being detected: Japanese lunch box (blue), Japanese rice ball (green), potato chip (yellow), and Japanese fried chicken (red).

Fig.8 RGB frame of the depth camera (Left) and extracted point cloud visualization (Right).

Fig.9 shows the extracted point cloud of a Japanese rice ball (onigiri) as seen from different perspectives. The left column shows the RGB images and the right column shows their corresponding point cloud information. The green cluster represents the point cloud of the detected object (onigiri) and the white cluster represents the depth information of the surroundings (plate, table). The point cloud information of each object is streamed in real-time as the camera raw data.

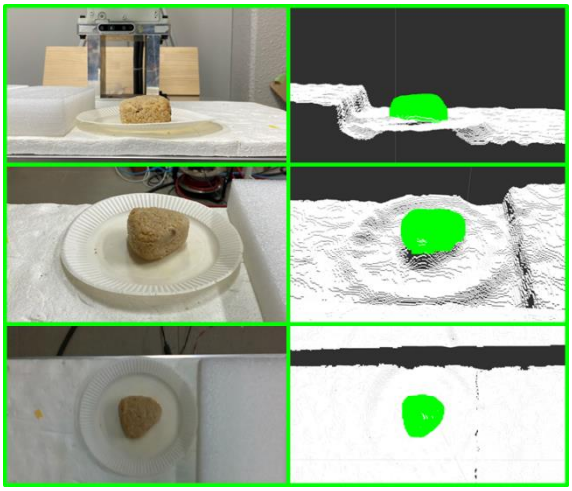

Fig.9 Japanese rice ball point cloud visualization

3.3 Extraction Method Comparison

Comparing between the two methods of point cloud extraction, the first extraction method shows a faster processing time as compared to the second extraction method. The processing time comparison of the first thousand frames is shown in Fig 8. Measured using *timeit* Python library, the average processing time of the first and the second extraction method are 0.149 and 0.211 seconds (6.71 and 4.74 fps), respectively. This translates to about a 1.4x faster processing time for the first extraction method. It is worth noting that although the second extraction method has worse performance, it is easier to understand and implement. Furthermore, the point cloud extracted using the second method retains the original RGB information of the detected object which could be beneficial for other types of analysis such as quality assessment or shelf life estimation based on color information.

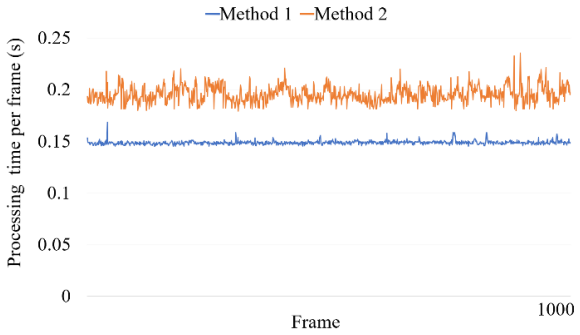

Figure 8. Comparison of processing time between two methods of point cloud extraction.

4. CONCLUSION

In this study, we perform instance segmentation on Japanese food images dataset using Cascade Mask R-CNN deep learning model and we also extract the corresponding point cloud of the detected objects using only one stereo depth camera mounted on a robotic arm system. We present and compare two methods for extracting point cloud information using different subscription topics, which shows trade-offs in performance vs. ease of implementation. The resulting output data from our process are in a form of ROS messages that are published continuously in the robot system network. The data should help enable robotic arm to manipulate and perform pick-and-place task on food object that varies in size, shape, and texture. Additionally, the output data can also be applied to other areas of food automation involving weight or calorie estimation of food objects.

REFERENCES

[1] Phil Hoden, "Automation in the food industry", https://www.newfoodmagazine.com/article/5424/automation-in-the-food-industry/, 2011.

[2] Chen, K., Pang, J., Wang, J., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Shi, J., Ouyang, W., et al.: "Hybrid task cascade for instance segmentation". In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019.

[3] Kai Chen, Jiangmiao Pang, Jiaqi Wang, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. "mmdetection". https://github.com/open-mmlab/mmdetection/, 2018.

[4] Zhaowei Cai and Nuno Vasconcelos. "Cascade r-cnn: Delving into high quality object detection". *In IEEE Conference on Computer Vision and Pattern Recognition, 2018.*

[5] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. "Mask r-cnn". *In IEEE International Conference on Computer Vision, 2017.*

[6] SIA5F 7-Axis Articulated Arm https://www.motoman.com/en-us/products/robots/industrial/assembly-handling/sia-series/sia5f/

[7] Intel RealSense Depth Camera D435i datasheet https://www.intelrealsense.com/wp-content/uploads/2020/05/Intel-RealSense-D400-Series-Datasheet-May-2020.pdf

[8] Nvidia GeForce RTX 2080 Ti Graphics Card https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080-ti/

[9] COCO – Common Objects in Context http://cocodataset.org/

[10] COCO Annotator https://github.com/jsbroks/coco-annotator/

[11] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. "Automatic differentiation in pytorch". In *Advances in Neural Information Processing Systems Workshop*, 2017.