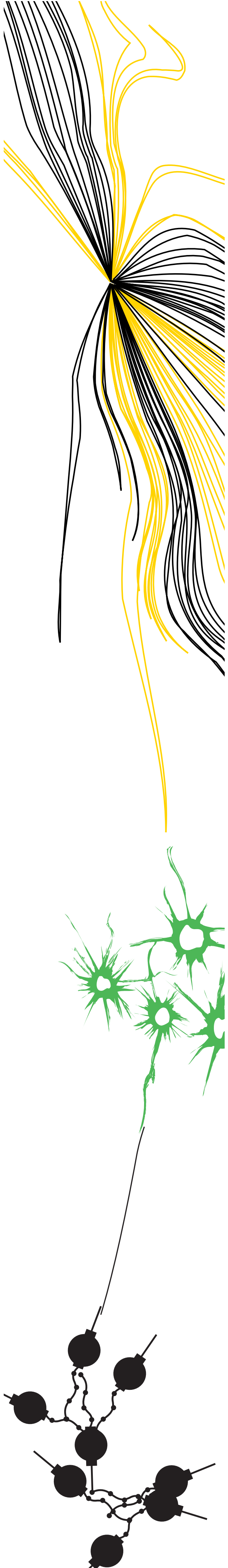


ROBOTICS AND MECHATRONICS

ROBOTIC BIN-PICKING PIPELINE FOR CHICKEN FILLETS WITH DEEP LEARNING- BASED INSTANCE SEGMENTATION USING SYNTHETIC DATA

L.M. (Marissa) Jonker

MSC ASSIGNMENT

Committee:
dr. ir. L.J. Spreeuwers
dr. ir. W. Roozing
dr. N. Strisciuglio

May, 2023

012RaM2023
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE
UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

DMB

DATABASE MANAGEMENT
AND
BIOMETRICS

.2854

ROBOTIC BIN-PICKING PIPELINE FOR CHICKEN FILLETS WITH DEEP LEARNING-BASED INSTANCE SEGMENTATION USING SYNTHETIC DATA

Marissa Jonker

MSC ASSIGNMENT

Committee:
dr. ir. L.J. Spreeuwers
dr. N. Strisciuglio
dr. ir. W. Roozing

May, 2023

2023DMB0005
Data Management and Biometrics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Robotic Bin-Picking Pipeline for Chicken Fillets with Deep Learning-Based Instance Segmentation using Synthetic Data

Marissa Jonker

***Abstract*—In the food processing industry, automation is getting more common for purposes such as increased food quality and compensation of worker shortages. An automation task such as gripping is challenging due to deformation of the objects. Additionally, in order to manipulate food in a specific manner, it is important to know the location and orientation of the food object. Due to these deformation and location problems, automation of tasks such as bin-picking food objects is a difficult challenge. Existing automated bin-picking methods for food objects lack environmental awareness and the ability to recreate the scene and objects in 3D in order to manipulate them and avoid collisions. In this research, we present a robotic pipeline for bin-picking of chicken fillets. The individual chicken fillets are detected using instance segmentation via deep learning. The instance segmentation model is trained on synthetically rendered images of chicken fillets. Current methods for synthetic data generation only use rigid body simulations, whereas we also simulate the deforming physics on manually created 3D models. Additionally, the path planning is based on a 3D reconstructed environment, using depth data from an RGB-D camera. The tests on the instance segmentation model with real data yield a bbox and mask AP@50:5:95 of 0.74 and 0.68, respectively. Finetuning the model with a small real dataset increases the APs to 0.82 and 0.78, respectively. Tests on the full pipeline show a planning success rate of 0.73, and an execution success rate of 0.81. We show that automation of bin-picking for chicken fillets using synthetic data is a realistic prospect.**

A supplementary video showcasing the pipeline can be found on https://tinyurl.com/5n8v8uf7.

I. Introduction

Food processing automation is being implemented for benefits such as increase of food quality, production efficiency and waste reduction. Furthermore, using robotics in food processing prevents workers from having to work in unpleasant environments, such as refrigerated rooms [1], [2]. Additionally, as there may not be enough qualified people to work in the industry, automation could compensate for this shortage [3]. The tasks in food processing automation range from standard automation tasks such as bin-picking, packaging, palletising and inspection, to food-specific tasks like deboning and cutting meat [4]. Bin-picking of food products in particular is a challenging task due to the varying shapes, textures, deformability and poses of the food. Locating different instances of food in a scene is a complicated task, for which nowadays often deep learning is utilised [5]–[7]. In order to have effective deep learning algorithms, large training datasets are required. Especially for instance segmentation making such a dataset is a tedious and difficult task, as every instance in each image has to be manually annotated. Consequently, there is a rise in interest and research in using synthetically rendered datasets [6], [8] and the transferability of the trained models to the real world [9].

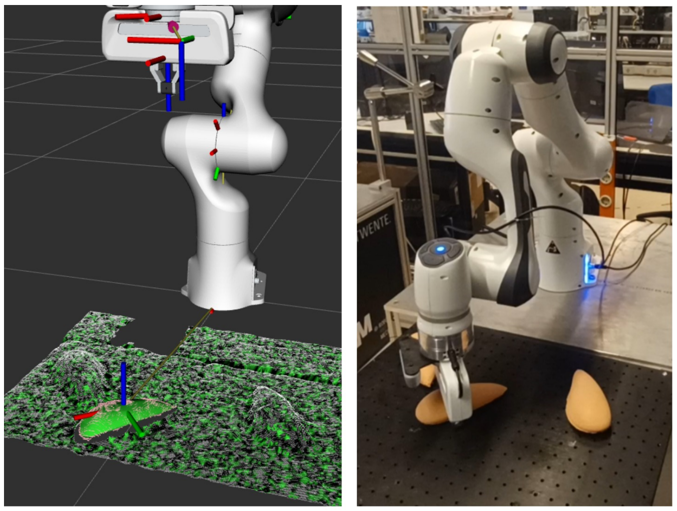


Fig. 1: Left: Visualised 3D camera measurements with convex hull and frame of detected chicken fillet. Right: Real bin-picking execution on the detected chicken fillet.

The objective of this paper is to present the design of a robotic system to perform bin-picking of a food object - specifically chicken fillets - placed in a tray or on a flat surface (Fig. 1). The purpose of this would be to e.g. transfer the chicken fillets from a bin to a conveyor belt, or into a container. When there are multiple chicken fillet instances in a pile, one has to be selected for grasping, meaning the individual instances should be identified. The next steps are determining the grasping affordance, path planning and executing the path accordingly, grasping the object, and using a robot arm to perform manipulation to place the object in a specific pose, for e.g. aesthetic reasons. The choice of using chicken fillets as objects introduces some challenges: they are deformable and have a homogeneous surface and smooth shape, which makes it difficult to distinguish multiple instances. The deformability and slimy surface of chicken fillets cause another issue, which is also encountered in the grasping phase; the chicken fillet should be grasped well enough such that it can be lifted, but it should not be dropped or damaged.

The recent work by Ummadisingu et al. [10] addresses a similar problem, and shows a system for food grasping in a cluttered scene using a synthetically generated dataset. The research bases its grasping on a fitted ellipse on the 2D segmented instances. A grasp is selected by checking the median heights (obtained with an RGB-D camera) of the objects to avoid gripper-object collisions with surrounding objects. In contrast, our approach uses the depth data from the RGB-D camera more extensively. We utilise 3D point cloud data to construct convex hull representations of objects to be grasped. Additionally, we create a mesh of the rest of the scene for path planning with collision avoidance. Consequently, we do not only avoid gripper-object collisions, but also gripper-scene, and object-scene collisions.

In this work, we present a full pipeline from instance segmentation to path planning and execution. Starting with the instance segmentation, we utilise the existing Mask R-CNN network proposed by He et al. [11]. We develop automated image dataset generation of chicken fillets including deformability, by using physics simulation. By masking the depth data from an RGB-D camera with the predicted instance masks, 3D objects of the chicken fillets and the scene are created, which in turn are interpreted for path planning. Then the grasp affordance is computed along the second principal axis of the 3D chicken fillet and based on these principal axes we place the chicken fillet in a desired pose. Although in this work we do not explicitly address the problem of grasping a deformable and slimy object regarding the friction, or rather the lack thereof, the work allows for modifications such as exchanging the grippers for ones more suitable for grasping chicken fillets. To evaluate our method, we conduct 30 tests on mock-up chicken fillets. The instance segmentation model is evaluated on a real dataset to get an indication of the transferability [9].

The main contributions of this study are:

- A full robotic bin-picking pipeline with collision avoidance based on 3D scene reconstruction. We mask the depth data from the RGB-D camera with the instance segmentation mask to obtain a 3D representation of the chicken fillet, which is interpreted for path planning. With the additional 3D reconstruction of the scene, the grasp and path planning are computed accordingly to avoid collisions between both the robot arm, grasped object, and scene.
- An expansion on the data generation algorithm proposed by Karoly and Galambos [8], utilising deformable object physics to incorporate the deformability of chicken fillets, and applying domain randomisation [9].
- The synthetic dataset and with it a retrained Mask R-CNN network [11], based on chicken fillet models. Results both on synthetic and real-world chicken fillet images demonstrate transferability.

This paper is organised as follows: In Sec. II we summarise relevant literature. We discuss the data generation and instance segmentation model training in Sec. III. After this we elaborate on the pipeline in Sec. IV, where we describe the instance segmentation to 3D object point cloud generation, the scene reconstruction and the path planning. We discuss the experimental setup in Sec. V, and present the results in Sec. VI. We interpret and discuss the implementation and its results in Sec. VII, and finally we give conclusions and recommendations in Sec. VIII.

II. Related Work

A. Bin-Picking Algorithms

While industrial bin-picking has been around for several decades [12], it is still an actively researched topic [13]–[16]. Bin-picking algorithms usually consist of the following components: instance segmentation, grasp planning, path planning and execution. With the automation of the food processing industry, new bin-picking methods are introduced to account for the vulnerability of food [10], [17]. Important factors that arise with food objects is that they are deformable and easily damaged. Therefore, collision-awareness is desired. The method proposed by Lou et al. [16] addresses avoiding collisions between the robot arm and the bin. However, this work does not incorporate the object poses for specific placement, which in the food industry is essential (for e.g. placing a chicken fillet in a container). Contrary to this, we do implement object pose estimation in our pipeline. The work by Ummadisingu et al. [10] addresses collision avoidance between the grippers and food objects. This is achieved by a grasp filtering strategy, based on fitted ellipses to the 2D segmentation masks and the median height obtained with an RGB-D camera. In contrast, we reconstruct the scene in 3D, such that collisions between the entire robot arm, grippers, and the rest of the scene are avoided. Additionally, as the (to be) grasped object is also reconstructed in 3D, we prevent collisions between the object and other objects in the scene, which in turn allows for grasp and path planning according to what is in the scene.

B. Instance Segmentation

The instance segmentation component of a bin-picking pipeline determines whether an object is recognised and can thus be grasped. With instance segmentation, an instance mask per object instance is computed. Different from semantic segmentation, where *stuff* is segmented (e.g. table, chicken), instance segmentation segments all *things* (chicken1, chicken2, etc.) [18]. While panoptic segmentation [19] does both (table, chicken1, chicken2, etc.), we deem instance segmentation sufficient for the purpose of this research, as we assume a simple scene: a few pieces of chicken fillets positioned on one flat surface, so everything that is not considered an instance is automatically background. Instance segmentation can be performed via clustering methods [13], [17], or by using deep learning [7], [11], where a neural network is trained on images with corresponding instance masks. As in the food industry many types of food are processed, we consider it more suitable to employ deep learning, such that with the instance segmentation, we can also perform object detection, which provides flexibility for different object classes, or for different tasks. The model we use is the popular Mask-RCNN as proposed by He et al. [11].

C. Synthetic Dataset for Network Training

Training on synthetically generated datasets is a powerful tool to bootstrap segmentation networks for object detection and instance segmentation [9]. Large training sets including instance masks can be generated automatically. There are a few works which look into creating such datasets, with a modeling program such as Blender [6], [8], but these works only use primary shapes. Existing work [10] uses realistic 3D food models for dataset generation. However, these are simulated as rigid bodies, whereas we include deformable object physics to incorporate the deformability property of the chicken fillets, making the rendered images more realistic and reducing the need for many different object models. By using domain randomisation, the generalisation of a model to the real world can be done by only using synthetic data [9], [20]. Domain randomisation implies that for example the number of objects, the textures, the lights, and the camera viewpoints are randomised. We use the same principle to generate a synthetic dataset. However, finetuning on a real dataset is shown to improve deep learning networks [6], [9]. Therefore, we also finetune on a small real dataset.

D. *Perception and World Modelling*

Having a camera above the scene, as in [10], [13], is useful for our case, since we want to grasp a piece that is easy to reach, which would be one on top of the pile and is not occluded or hindered by other pieces. A straightforward way to find such a piece is by looking at the scene from above, especially with an RGB-D camera. In many bin-picking implementations eye-*in*-hand is used: the camera is attached to the end-effector [21], [22]. The camera could also be static, always looking in the same direction [3], [10]. Since multi-view scan of the scene increases the accuracy of the generated 3D data [23] and deals with object occlusions, we attach the RGB-D camera to the end-effector of the robot arm for the intended purpose of implementing multi-view scene reconstruction. Since we want to take into account the surroundings and avoid object collisions, resulting in for example damaged chicken fillets, we use the RGB-D data to reconstruct the scene in 3D. With 3D representations of both the chicken fillets as well as the scene, the path is planned accordingly, such that collisions between the objects are avoided.

E. *Path Planning*

1) Object pose: A straightforward method for object pose detection is using template matching [21]. Template matching requires keypoints, so this method is ideal for packaged food products such as soup cans or cookie boxes, which include many keypoints due to both the printed labels and object shape. However, template matching is unsuitable for chicken fillets, due to deformation and the lack of features. Another method to obtain the object pose is by using deep learning. Xiang et al. [24] introduce a neural network PoseCNN, with which the pose transformation between the camera frame and the object frame can be determined. However, for deformable objects it is difficult to define a pose, hence the poses of the chicken fillets after simulating the deforming physics are not considered to be reliable training data. Thus, we choose not to use this network. Pohl et al. [13] show that with Principal Component Analysis (PCA) it is possible to grasp objects. Assuming the object has a uniform mass distribution, the principal axes are found by performing PCA on the points in the segmented object point cloud. We obtain the segmented point cloud by translating the instance segmentation masks similarly to [7], where the instance mask is projected on the depth data from an RGB-D camera. This way, the instance can be distinguished in a point cloud. The principal axes define the body frame of the chicken fillet in its current shape. This means that the principal axes of the chicken fillet might point in a different direction depending on how the object is deformed. However, due to the nature of PCA, we assume that the resulting body frame describes the object pose sufficiently for grasping.

2) Grasp affordance and Path Planning: Studies show several methods for grasp affordance generation, such as deep learning methods [25], [26], using geometric properties [10], [27], [28], or template matching [21]. Pohl et al. [13] compute the grasp affordance along the second principal axis found by PCA. We opt for the latter method due to its simplicity and the way it utilises the already obtained results from pose estimation. The principal axes are used in combination with MoveIt! Task Constructor (MTC), a framework to plan the motion for robotic manipulation, as presented by Gorner et al. [29]. We choose to let the grasp affordances be computed using MTC, in accordance with the principal axes.

III. Instance Segmentation

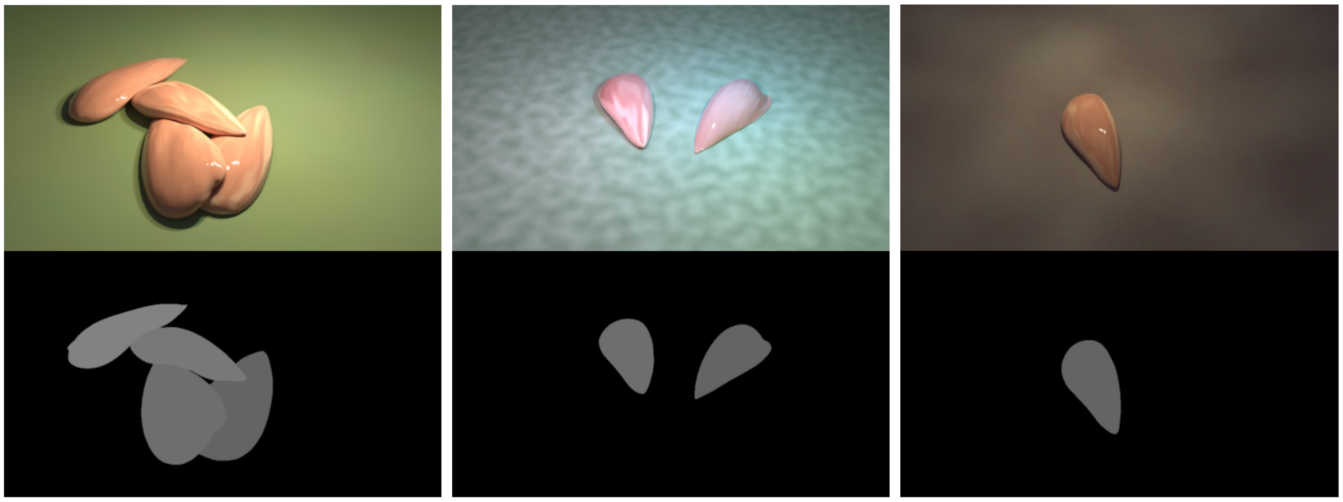
In this section, we elaborate on the implementation of the instance segmentation part of the pipeline, highlighting the synthetic dataset for training.

A. *Synthetic Dataset*

To generate the synthetic dataset, we use Algorithm 1 in Blender, with the parameters from Table I. In total, we render a dataset of 1500 images (300 scenes, 5 images of each scene). Example images for different parameter values are shown in Fig. 2. Before generating the dataset, we create two 3D models of a chicken fillet in Blender as shown in Fig. 3. The shape of the models are created by sculpting *icosphere* meshes along example images of chicken fillets. For later simulation of the physics we prefer to use an icosphere over

Algorithm 1 Generating synthetic training data

```
1:  for 300 scenes do
2:      Remove objects from last scene
3:      Change texture and colour of plane to N and SC
4:      for F do
5:          Spawn one of the two fillet models
6:          Scale, rotate model
7:          Assign index pass
8:      end for
9:      Bake object physics
10:     for L do
11:         Spawn light
12:         Change light colour to LC
13:         Change light intensity to LI
14:     end for
15:     for C do
16:         Spawn camera
17:         Point camera at scene
18:         Render image
19:     end for
20: end for
```



(a) Four overlapping fillets, three light sources. **(b)** Two fillets, three light sources. **(c)** One fillet, one light source.

Fig. 2: Example images and respective instance masks of the synthetic dataset.

TABLE I: Synthetic dataset render parameters

Parameter	Value	Parameter name
# fillets	1-5	F
# light sources	1-5	L
Light colour	$RGB \sim U(0,1)$	LC
Light intensity	1100-1300	LI
# cameras	5	C
Surface texture	Fractal Perlin noise	N
Surface color	$RGB \sim U(0,1)$	SC

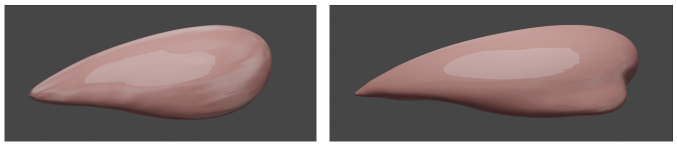


Fig. 3: Chicken fillet 3D models.

a uv-sphere due to the distribution of vertices. The vertices in icospheres are evenly distributed, whereas in uv-spheres they are centered around an origin point. As chicken fillets have side lobes in some cases, we let one of the models represent this. The surface colour and texture of the fillets are based on example images of real chicken fillets. To increase the realism of the surface, we mimic lines of fat by adding wave textures of a lighter colour, with a random phase offset for generalisation. We apply physics modifiers, such that the 'deformable object' physics can be simulated. For our purpose of imitating the deformability of chicken fillets, we use the *cloth* modifier, including internal springs. With this modifier, we simulate chicken fillets falling on top of each other, and deforming as such. To obtain the instance masks, we use the *cycles* render engine, and assign an index pass (integer value) to each chicken fillet in the scene. When training the network, the generated masks are converted to a binary mask for each object separately. For the scenes, we assume that the fillets are positioned on a flat surface, without any other objects in the scene. By applying domain randomisation (Table I) we generalise the model, making it more suitable for real life scenarios [30]. We give the surface the chicken fillets are placed on a random noise texture and color, and also vary the number of chicken fillets, number of light sources and their properties, as well as the camera positions. Per generated scene, we spawn five virtual cameras, each taking a picture of the scene which is added to the dataset.

B. Training Details

We train the commonly used Mask-RCNN instance segementation model [11] with ResNet-50-FPN backbone from [31], which is initialised with pretrained weights from the Microsoft COCO dataset [32]. We use 1280 images to train, with a learning rate of 10^{-5}, and weight decay of 10^{-8}, both heuristically determined. Furthermore, we use a batchsize of 6, for 60 epochs.

As suggested by Park et al. [6], finetuning the model on a real-world dataset increases the performance on real data. Therefore, we finetune the model, using a training set of 14 real images. The real data are manually annotated using Labelme [33], which yields polygon data describing the masks. The polygon data are converted to binary masks prior to training and testing. Because we only want to finetune, we choose a smaller learning rate of 10^{-8}, and a weight decay of 10^{-9}. We finetune the network for 10 epochs.

C. Evaluation Metrics

As input, the Mask-RCNN model expects an RGB image, and outputs predicted binary masks, confidence scores and bounding boxes. The trained networks are evaluated using the standard average precision (AP) metric [34], which is calculated using the Intersection over Union (IoU) of the predicted and ground-truth masks:

$$IoU = \frac{\text{Area of intersection}}{\text{Area of union}}. \quad (1)$$

The IoU indicates whether a predicted mask is a false positive (FP) or a true positive (TP), depending on a threshold between

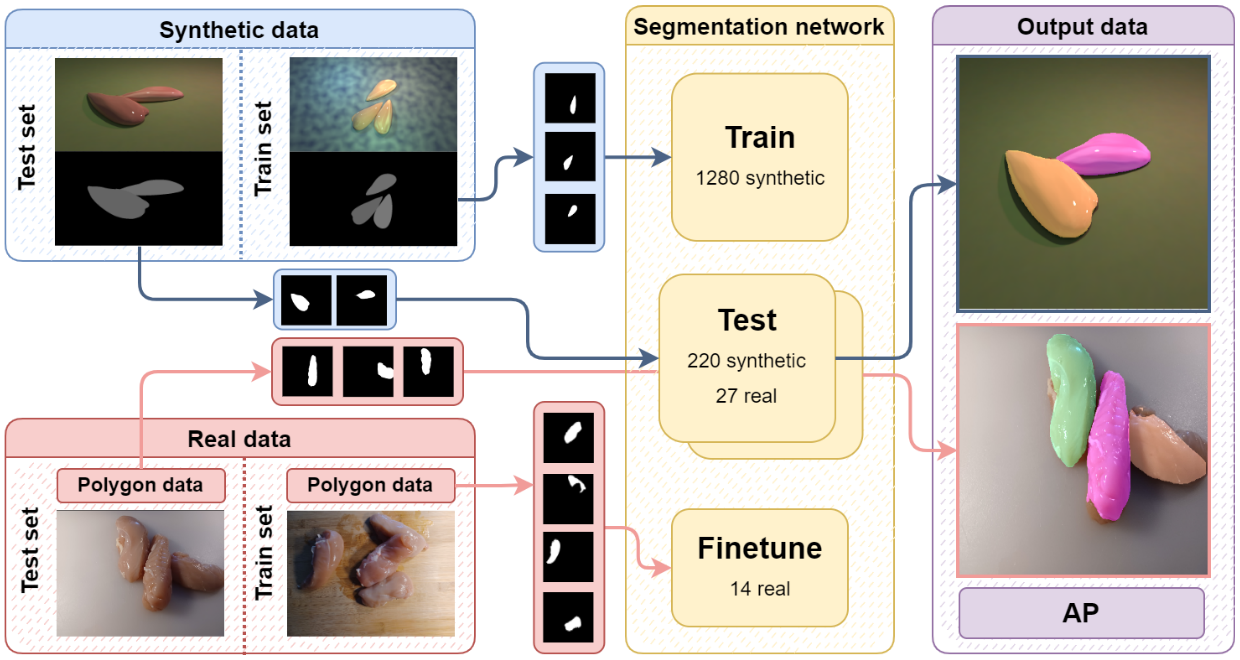


Fig. 4: Instance segmentation model training and testing process. Initially, the Mask-RCNN network is trained on synthetic data. Afterwards, finetuning is done using real data. The testing of both versions of the model is done on the same test sets of synthetic data and real data. The model outputs confidence scores, bounding boxes, and binary masks, with which we calculate the average precision (AP). In this figure we overlay the predicted masks on the input images.

0 and 1 (an IoU of 1 indicates the predicted mask is identical to the ground truth mask). We count the FPs and TPs for a range of IoU thresholds: from 0.5 to 0.95 with steps of 0.05. The APs are then obtained by estimating the areas under the respective precision-recall curves using 11-point interpolation per IoU threshold, as shown by Padilla et al. [34]:

$$AP = \frac{1}{11} \sum_{R \in \{0, 0.1, \dots, 1\}} P_{\text{interp}}(R), \tag{2}$$

where

$$P_{\text{interp}}(R) = \max_{\tilde{R}: \tilde{R} \geq R} P(\tilde{R}) \tag{3}$$

and $P(\tilde{R})$ the precision-recall curve. We evaluate the network by looking at the APs for IoU thresholds 0.5, 0.75, and averaged over the complete range, commonly denoted as AP50, AP75, and AP@50:5:95, respectively. The APs for both the bounding box (bbox) and object mask are computed. This is performed on the synthetic dataset (220 images) and on a small real dataset (27 images). Fig. 4 shows the process of which data is used for training and testing the instance segmentation model.

IV. Bin-Picking Pipeline

Fig. 5 shows an overview of the full bin-picking pipeline. In the following sections we elaborate on each block shown in the image. The pipeline is implemented using communication between ROS (Robot Operating System) nodes, of which the structure can be seen in Fig. 6.

A. Perception

We start the pipeline by capturing both an RGB image, as well as a depth image, using an Intel RealSense D435i. From the RGB image, we obtain the instance masks by using the trained Mask-RCNN network described in Sec. III. We select which of the objects to pick by using the predicted 2D instance masks confidence scores, as well as the depth data, and the distance of each detected chicken fillet to the frame centre. We assume that the object which is easiest to pick and least likely to be covered by another instance is one on top of the pile, hence we prefer the instance with the smallest median depth. We weigh the three variables, and select the instance based on the maximum value:

$$\text{sel. instance} = \arg\max_{p} \left(0.1 c_p + 0.8 {d_p}^{-1} + 0.1 {f_p}^{-1}\right), \tag{4}$$

where for each predicted instance p, c_p is the confidence score, d_p the median depth and f_p is the sum of the x and y distances of the chicken fillet mask centre coordinates to the frame centre. Once an instance mask is selected, the corresponding masked depth image is further processed. The inverse of the selected mask is taken as a representation of the rest of the scene and is consequently used to mask the depth image.

B. Point Cloud Processing

From the masked depth images, we compute the point cloud using existing conversion functions for the RealSense. Note that the depth values in the depth image are with respect to the camera, thus the point cloud coordinates are also with respect to the camera. For the robot control later on, it is desired to have these coordinates in an inertial frame, i.e. the

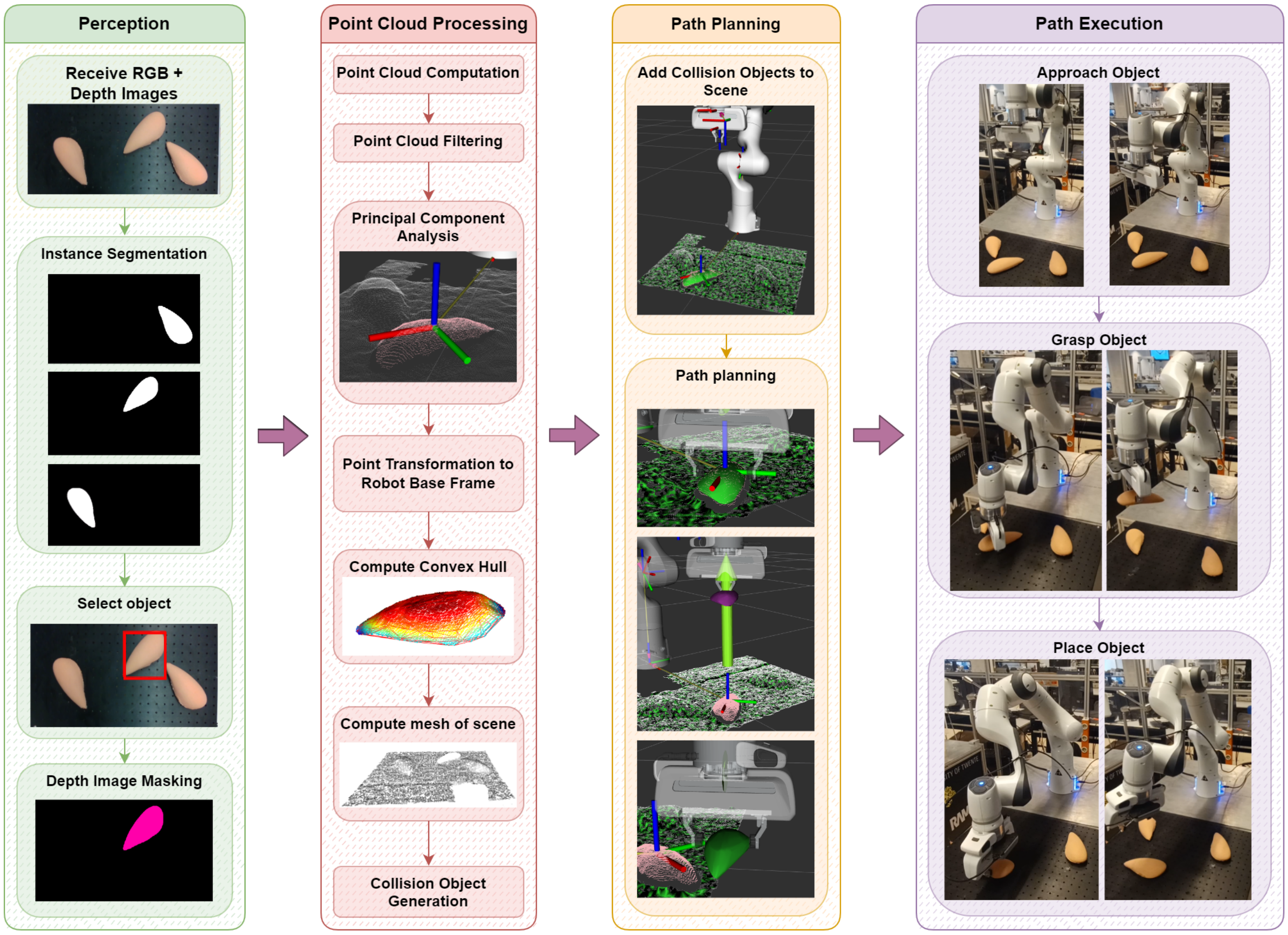


Fig. 5: The bin-picking pipeline, separated into different main tasks.

robot base frame, so we transform the points to this frame prior to further processing. Doing so also allows us to easily implement multi-view reconstruction later on, as all points are transformed to the same inertial frame. Obtaining the transformation between the frames is done using the `/tf2` node (Fig. 6). The transformation is given in an xyz translation

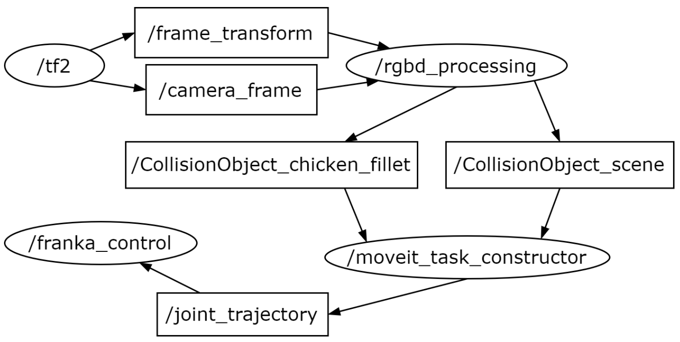


Fig. 6: ROS computation graph. The `/rgbd_processing` node performs the steps *Perception*, and *Point Cloud Processing* in Fig. 5, using frame transforms from the `/tf2` node from the ROS tf2 library. The `/moveit_task_constructor` node performs the *Path Planning*, and `/franka_control` [35] the *Path Execution*.

$\boldsymbol{t} \in \mathbb{R}^3$, and a rotation represented by a quaternion $\boldsymbol{q} \in \mathbb{R}^4$. We define the point cloud with respect to a certain frame $\boldsymbol{P}_{<\text{frame}>}$ as an $M \times 3$ matrix, where M denotes the number of points. A single point $\boldsymbol{p}_m \in \mathbb{R}^3$ is then defined as the m^{th} row of $\boldsymbol{P}_{<\text{frame}>}$ where $m = 1, 2, \ldots, M$. To transform the points from the camera frame to the inertial frame, for each m^{th} point in the point cloud we use

$$\boldsymbol{p}_{\text{inertial},m} = (\boldsymbol{q}\boldsymbol{p}_{\text{camera},m}\boldsymbol{q}_{\text{inv}}) + \boldsymbol{t}, \tag{5}$$

where $\boldsymbol{q}$ and $\boldsymbol{q}_{\text{inv}}$ are the obtained quaternion and its inverse. Note that to make the multiplication possible (as $\boldsymbol{q} \in \mathbb{R}^4$ and $\boldsymbol{p} \in \mathbb{R}^3$) we append the point cloud $\boldsymbol{P}_{\text{camera}}$ with a column of 0s. The extra value is then removed prior to the addition.

To ensure that there are no outliers present to influence the PCA and convex hull computation, we apply voxel downsampling with a voxel size of 0.001 and statistical outlier removal on the point cloud of the chicken fillet (40 neighbours and a standard deviation of 3). Then we compute the body frame of the selected chicken fillet by PCA on the filtered point cloud. We determine the convex hull of the chicken fillet point cloud using Open3D [36]; it is necessary to convert the chicken point cloud to an actual object which can be recognised in MoveIt! (`moveit_msgs/CollisionObject`), that in turn computes the grasps and the manipulation path. The remainder of the point cloud (i.e. the scene) is first downsampled to a voxel size of 0.005, after which we compute a triangle mesh, which makes up another `CollisionObject`, such that the path planner can plan around this.

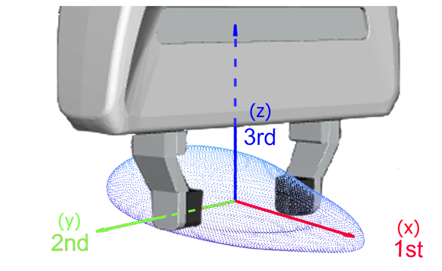


Fig. 7: Illustration of grasp affordance based on the principal axes.

C. *Path Planning and Execution*

The path planning is done in MTC. We publish the detected chicken fillet instance as well as the remainder of the scene as `CollisionObject` topics from the `/rgbd_processing` node, after which the `/moveit_task_constructor` node subscribes to these. The `CollisionObject` message contains all point positions, the vertices of the mesh, as well as the calculated body frame from the PCA. Then according to the calculated body frame of the chicken, the grasp poses are computed as in Fig. 7: the grippers closing along the second principal axis, while approaching along the third principal axis of the chicken fillet. As also indicated in the illustration, we refer to the first principal component (red) as the x axis, the second (green) as the y axis, and the third (blue) as the z axis. We continuously use the same colours for these axes in the remainder of this paper.

MTC is given a constant placement pose, with respect to the base frame. Avoiding collisions is taken care of by the `CollisionObjects` in MTC, which are not allowed to collide with each other, and a possible collision in the movement calculations rules out that movement.

After the path planning is done, we have a plan which has to be executed by the robot arm. We do this by using position control with `/franka_control` in ROS, as MTC outputs the joint positions for the whole path.

V. Experiment Setup

In this section we show the experiment setup and list the materials used for testing the pipeline. Additionally, we show multiple test cases and evaluation metrics.

A. *Materials*

The following materials and software are used:

- Intel RealSense D435i as RGB-D camera. The camera is attached to the Franka Emika Panda robot end-effector (eye-in-hand), facing in the same direction as the gripper fingers, as can be seen in the experiment setup in Fig. 8.

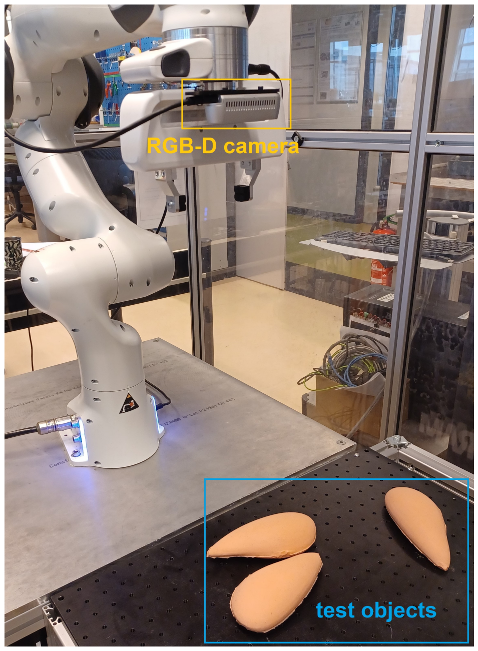


Fig. 8: Experiment setup.

- The image generation and network training are done on an external GPU on a cluster (16 GB RAM), using CUDA 11.6 and Blender 3.2.1 with Python 3.8.10. The network training is done using similar hardware, using Pytorch.
- Franka Emika Panda 7-DOF robotic arm with its standard parallel finger grippers.
- The pipeline from image acquisition to path planning is implemented with ROS2, MoveIt!2 [37] and MoveIt! Task Constructor [29].
- Silicone chicken fillet mock-ups made from Ecoflex 00-10 and 00-30 poured into a 3D-printed mould of the chicken fillet model (Fig. 8).

B. *Test Cases*

We consider three types of cases for testing, from easy to difficult: I): one chicken fillet, II): three non-overlapping chicken fillets, and III): three touching/overlapping chicken fillets. The three cases are shown in Fig. 9. We define a test as follows: one test case is chosen within which the chicken fillets are placed in arbitrary poses. Then the pipeline is executed once for each test. This is done five times for the three cases.

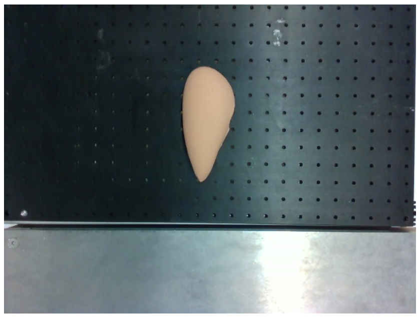
C. *Evaluation Metrics*

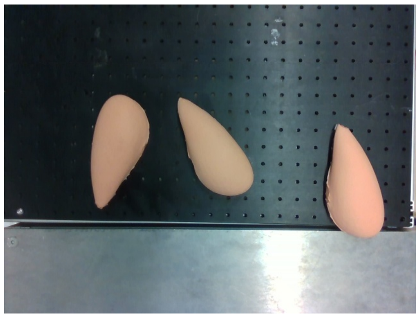
We evaluate the full pick-and-place pipeline based on success rate of the path planning and execution. We define these respectively as r_p and r_e:

$$r_\mathrm{p} = \frac{\text{\# successfully planned paths}}{\text{\# total planning attempts}}, \tag{6}$$

$$r_\mathrm{e} = \frac{\text{\# successfully executed paths}}{\text{\# total execution attempts}}. \tag{7}$$

If a planning attempt succeeds, this will count towards the execution attempts. An execution attempt is deemed successful if the chicken fillet is placed correctly in the placement pose. The placement pose is constant for each test.


(a) Case I: one chicken fillet.


(b) Case II: Three non-overlapping fillets.

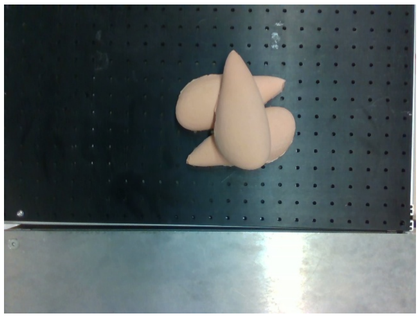

(c) Case III: Three overlapping chicken fillets.

Fig. 9: Three test cases for testing the pick-and-place system.

VI. Results

In this section we present the results of the instance segmentation, as well as the performance of the bin-picking pipeline.

A. Instance segmentation

The instance segmentation model trained on only synthetic data, and later finetuned with real data are validated. We test both models on synthetic and real images, and compare the two models to see to what extent finetuning improves the performance. Table II shows the AP scores for testing on the synthetic and the real datasets for both the train sets. For training on only the synthetic dataset, the mask AP@50:5:95 of testing on the synthetic data is 0.86, the bbox AP is 0.93. The model does not perform as well on real data, with a mask AP of 0.68, and bbox AP of 0.74. For the finetuned model, we see that both the bbox and mask AP@50:5:95 are lower for the synthetic test set, to 0.90 and 0.83 respectively. The performance on an IoU threshold of 0.5 is similar. However, the performance on the real dataset has improved, as there is

TABLE II: Test results of box and mask AP for both datasets. The AP scores on the real dataset are highlighted to show the improvement of finetuning on the instance segmentation model.

Train set	Test set	Input	AP@50:5:95	AP50	AP75
Synthetic	Synthetic	bbox	0.93	0.99	0.96
		mask	0.86	0.90	0.90
	Real	bbox	0.74	0.92	0.75
		mask	**0.68**	**0.81**	**0.71**
Synthetic + real	Synthetic	bbox	0.90	0.99	0.94
		mask	0.83	0.90	0.90
	Real	bbox	0.82	0.95	0.93
		mask	**0.78**	**0.90**	**0.81**

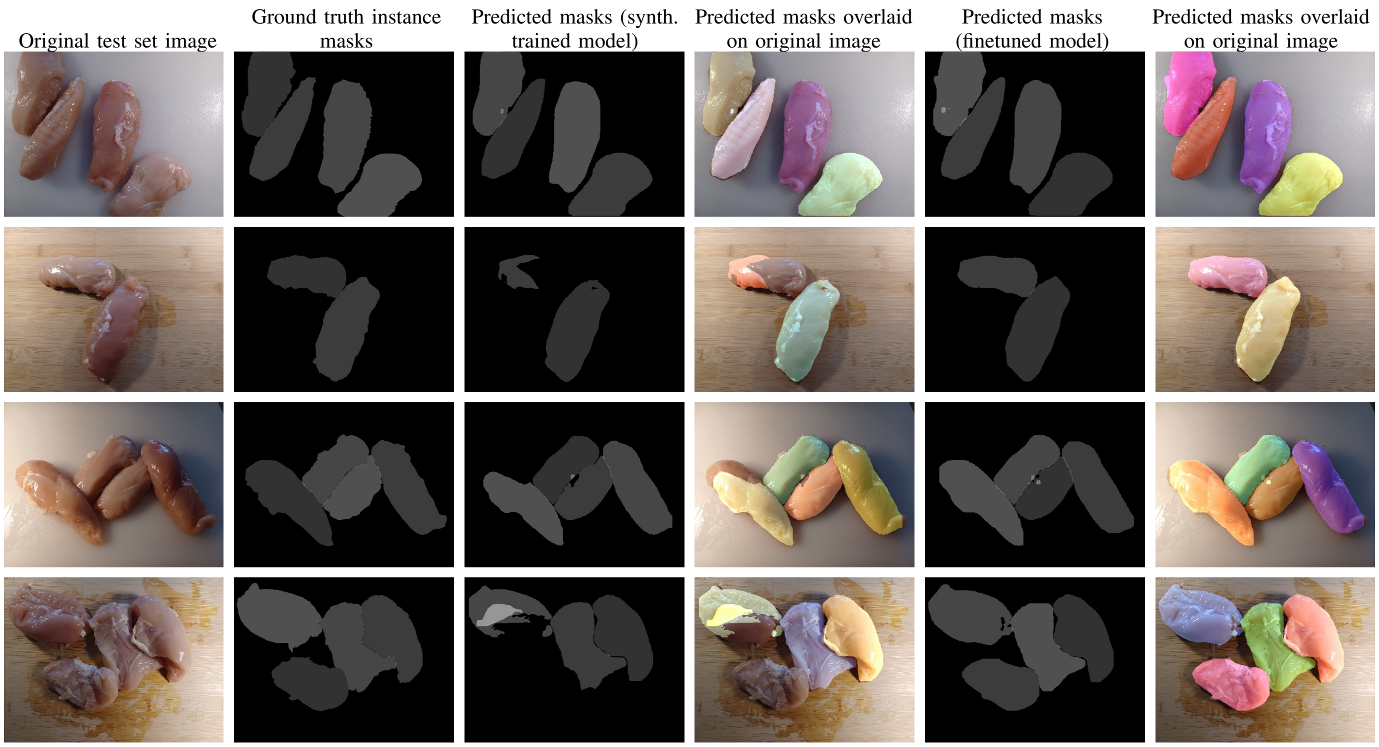

Original test set image
Ground truth instance masks
Predicted masks (synth. trained model)
Predicted masks overlaid on original image
Predicted masks (finetuned model)
Predicted masks overlaid on original image

Fig. 10: Instance segmentation test results. For each row from left to right: original test set image; ground truth instance masks; predicted masks from model trained on synthetic data; predicted masks overlaid on original image; predicted masks from finetuned model; predicted masks overlaid on original image.

an increase for the bbox and mask AP to 0.82 (+0.08) and 0.78 (+0.1), respectively. We see that for every IoU threshold the performance of the finetuned model on real data has increased, as expected. Fig. 10 shows a few examples of masks predicted by the two models on the same images. It can be seen here that the masks predicted by the finetuned model are more similar to the ground truth masks than those predicted by the synthetically trained model. It seems that primarily chicken fillets with notable lines of fat are segmented incorrectly. In these cases, we see that the predicted mask boundaries often lie on these lines. The model does not seem to have much problem with detecting the borders between different instances, even in most cases where there is overlap.

The results suggest that with the domain randomisation in generating the synthetic dataset, we are able to predict masks close to the ground truth for the majority of the test images. However, as the results show, we were not able to bridge the reality gap completely. As expected, finetuning the model with real images improved the model significantly.

B. Bin-Picking

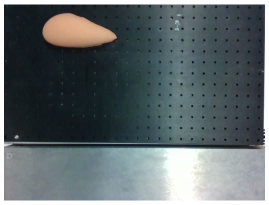
We proceed with the results of the full bin-picking pipeline. The tests for all three cases (single, multiple non-overlapping, and multiple overlapping fillets) are shown in subsequent rows of Fig. 11. We evaluate the pipeline on different cases and difficulties to test the robustness to various orientations, positions, overlap, and collision prevention. The results of the experiments are summarised in Table III. We find a planning success rate of $r_\mathrm{p} = 0.87$ and an execution success rate of $r_\mathrm{e} = 0.69$.

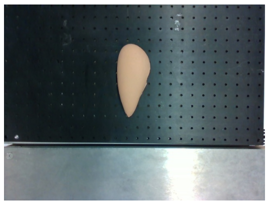
1) Test Case I: One chicken fillet: The tests for this case are shown in the top row of Fig. 11. There were two tests for which the path planning failed, shown in Figs. 11a and 11d. The failed planning attempts are due to the calculated collision between the Franka grippers and the convex hull of the chicken fillets, as shown in Fig. 12. We see that an inaccurate rotation and location of the centre of mass result in an impossible grasp affordance for the grippers, since we compute the grasping with the end-effector directly above the origin along the z-axis. In both the failed planning cases, the chicken fillet was positioned at the edge of the camera field of view. Furthermore, we find that the maximum distance between the gripper fingers is insufficient to grasp most fillets at their second principal axis, causing path planning failures due to collisions.

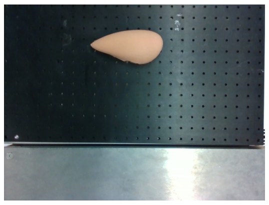
2) Test Case II: Multiple non-overlapping chicken fillets: The tests for this case are shown in the middle row of Fig. 11. For all tests a path was successfully computed. Although the test in Fig. 11i had a successfully executed motion, the grasp itself failed due to grasping the chicken fillet too high,

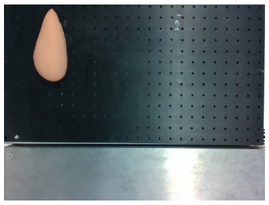
TABLE III: Results of the full bin-picking pipeline, indicating failure/success. Per case five tests were done.

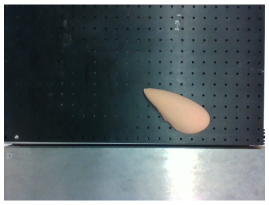
Case	Failed plannings	Grasp Executions	Successful Grasps	Failed Grasps
I): One Chicken Fillet	2	3	3	0
II): Multiple, Non-Touching	0	5	4	1
III): Multiple, Stacked	0	5	2	3

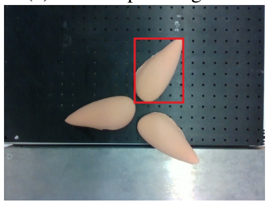

(a) Case I: planning fail.

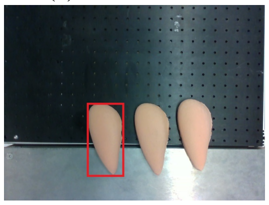

(b) Case I: success.

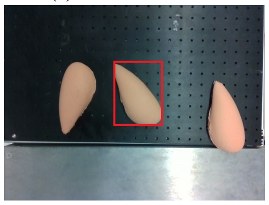

(c) Case I: success.

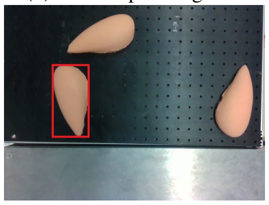

(d) Case I: planning fail.

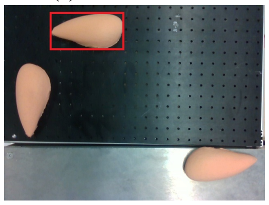

(e) Case I: success

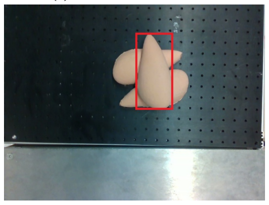

(f) Case II: success

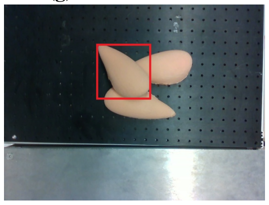

(g) Case II: success

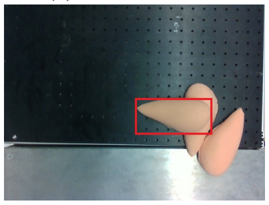

(h) Case II: success

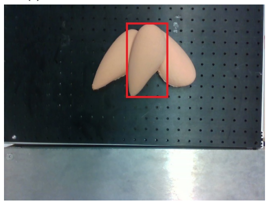

(i) Case II: execution fail


(j) Case II: success


(k) Case III: success


(l) Case III: execution fail


(m) Case III: execution fail


(n) Case III: success

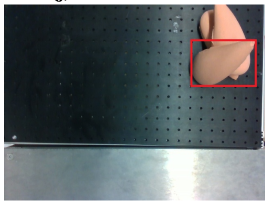

(o) Case III: execution fail

Fig. 11: View of the RGB-D camera for all tests. Whether the test is a failure or a success in indicated per image. For cases II and III the chicken fillets selected for grasping are outlined.

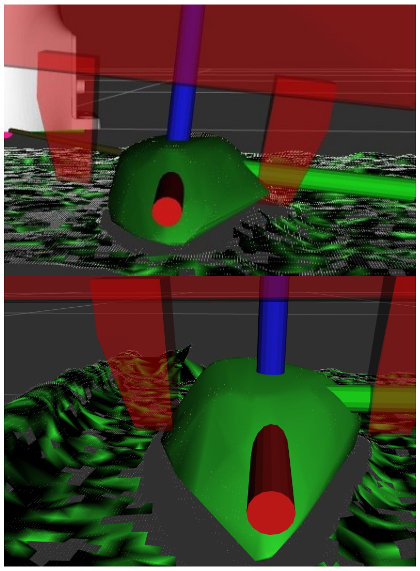


Fig. 12: Collisions between Franka gripper fingers and chicken fillet convex hull due to rotation (top) or location (bottom) of the centre of mass.

thus dropping it almost immediately. Inspecting the computed grasp in Fig. 13, we see that the grasp itself will not allow a successful pick-and-placement of the fillet, because the fillet is not rigid but deforms when the grippers close. We see that the calculated centre of mass is above the centre of the convex hull. This is because we calculate the centre according to the point cloud, of which the points are unevenly distributed, skewing the estimated centre upwards.

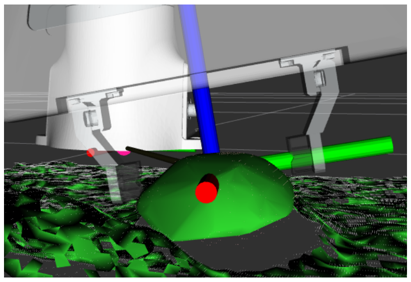


Fig. 13: Computed grasp for chicken fillet in Fig. 11i. The grasp is too high due to estimated rotation around x-axis, and inaccurate centre of mass.

3) Test Case III: Multiple overlapping chicken fillets: The tests for this case are shown in the bottom row of Fig. 11. For these tests, two out of the five were successful. Similarly to test cases (I) and (II), the calculated centre of mass for each fillet is higher than the true centre of the object, which is also shown in Fig. 14. While not being a problem in most tests previously, we see that for the chicken fillets in a slanted position, the grasping fails. For the tests in Figs. 11k and 11n - the successful executions - the fillets are positioned quite horizontally. The failed grasp for the test in Fig. 11m is shown in Fig. 15.

We find that for each of the fifteen tests the calculated centres of mass are too high compared to reality. This is due

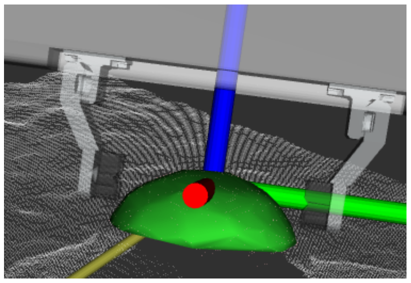


Fig. 14: Computed grasp for chicken fillet in Fig. 11l. The grasp is too high due to inacurrate centre of mass.

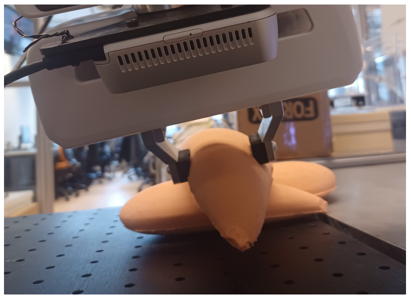


Fig. 15: Executed grasp for chicken fillet in Fig. 11m. The grasp is too high, resulting in the fillet to slip from between the finger grippers.

to the chicken fillet point clouds being incomplete; the parts of the fillets not visible to the RGB-D camera do not get converted into a point cloud. In most cases this did not result in a grasping failure. However, in case of the execution failures recorded in our tests, the cause was consistently grasping the fillets too high. The instance segmentation performed as desired.

C. Centre of Mass as Convex Hull Centre

Using only the point cloud to compute the centre of mass of the chicken fillets yields a centre of mass that is typically biased towards the direction of the camera, due to the distribution of point cloud data. To show that a lower estimated centre improves the grasping execution, we compute the centre of mass as the average of the extremes, starting with the z component:

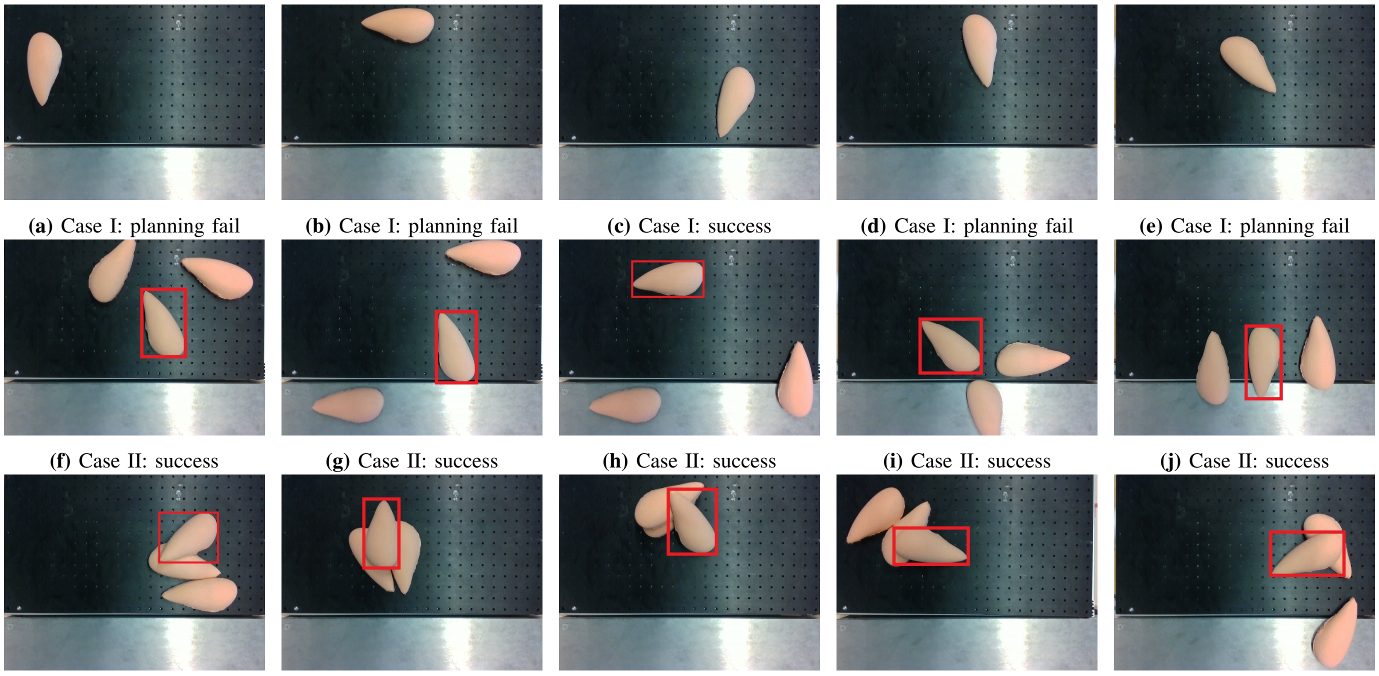
$$z_c = \frac{z_{max} + z_{min}}{2}. \tag{8}$$

Additionally, as we found in test case (I) that a slightly wrong y component of the centre of mass can result in a planning failure we recalculate it similarly to Eq. (8):

$$y_c = \frac{y_{max} + y_{min}}{2}. \tag{9}$$

As the point cloud has been complete in the x direction, we leave the centre of mass in this axis as the mean of the points.

We repeat the three test cases for a new set shown in Fig. 16. The results shown in Table IV demonstrate a slight improvement for cases (II) and (III). We find a total planning success rate of $r_{\mathrm{p}} = 0.73$ and execution success rate of $r_{\mathrm{e}} = 0.81$. We see that with the centre of mass being computed as the centre of the convex hull, the fillets are


(a) Case I: planning fail **(b)** Case I: planning fail **(c)** Case I: success **(d)** Case I: planning fail **(e)** Case I: planning fail

(f) Case II: success **(g)** Case II: success **(h)** Case II: success **(i)** Case II: success **(j)** Case II: success

(k) Case III: success **(l)** Case III: success **(m)** Case III: success **(n)** Case III: execution fail **(o)** Case III: execution fail

Fig. 16: View of the RGB-D camera for all tests with centre of mass calculation using Eqs. (8) and (9). Whether the test is a failure or a success in indicated per image. For cases II and III the chicken fillets selected for grasping are outlined.

TABLE IV: Results of the tests on the pipeline with the centre of mass y and z components calculated with Eqs. (8) and (9).

Case	Failed plannings	Grasp Executions	Successful Grasps	Failed Grasps
I): One Chicken Fillet	3	2	1	1
II): Multiple, Non-Touching	0	5	5	0
III): Multiple, Stacked	0	5	3	2

indeed grasped lower than during the first batch of tests. Consequently, the fillets are grasped successfully more often ($r_e = 0.81$ compared to $r_e = 0.69$). Similar to the first tests, the failed plannings are only present for case (I). Furthermore, we notice that for this case we have more planning failures. For case (II), we see that all grasp plannings and executions were successful. For case (III), two grasping attempts (Figs. 16n and 16o) did not succeed. These failures were due to the fillets being grasped too high. The computed grasp for one of the tests in Fig. 17 shows that the origin of the frame is in the middle of the chicken fillet convex hull (for the z-axis). However, the convex hull does not represent the full chicken fillet, due to self-occlusion and single view reconstruction of the fillet. Even though computing the centre of mass this way improves the grasping, the inaccurate frame rotation which was also present during the first batch of tests (e.g. Fig. 13) still remains.

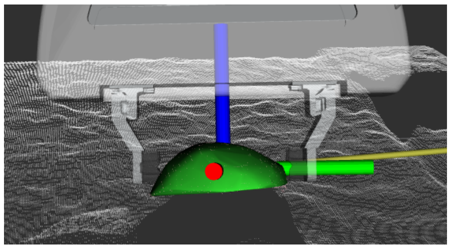


Fig. 17: Computed grasp for fillet in Fig. 16n. Grasp execution failed due to fillet grasped too high as consequence of self-occlusion.

VII. Discussion

The experiment results showed that the pipeline worked as intended, although we found that the gripper and utilisation of one viewpoint were the main drawbacks of the system.

A. Instance Segmentation

The instance segmentation with the model trained on only synthetic data can correctly predict chicken fillet instances, both synthetic and real, as was expected [9]. However, we find that there were some issues such as predicting one fillet as multiple, or not recognising a fillet at all. The model works quite robustly on simple shapes, as opposed to complex shapes: chicken fillets with lines of fat or extra parts of meat attached, resulting in rough shapes and non-smooth surfaces. We did not include such chicken fillet models in the synthetic dataset, so it is unsurprising that these are incorrectly segmented. As expected (and also proposed by Tremblay et al. [9] and Park et al. [6]), we find that finetuning the trained model with real images significantly improves the performance, as the mask AP increased from 0.68 to 0.78. The visual results in Fig. 10 show that the difficult cases wrongly predicted by the non-finetuned model are correctly predicted by the finetuned model. The model performs better on difficult shapes, but still experiences some difficulties distinguishing lines of fat from actual object borders. We expect altering the 3D models of the chicken fillets to include such lines more clearly will improve the performance of the model regarding the aforementioned issue. Additionally, if the shape of the chicken fillet 3D model is made more realistic, we expect the transferability of the model trained on a newly rendered dataset to be improved.

B. Bin-Picking Pipeline

Overall, we find from the tests that the pipeline performs as desired. We are able to successfully perform pick-and-placement on detected chicken fillets. The issues in the system, resulting in planning and execution failures, are due to incomplete point cloud data. The execution failures were due to the fillets being grasped too high. When estimating the centre of mass as the middle of the extreme values as opposed to the point cloud mean, we find that the grasping execution is more robust. However, grasping failed consistently for chicken fillets in highly inclined configurations. Estimating the centre of an object based on the average of the extremes will likely cause problems for different objects, or if more deformation is present. In our case, using only two points worked because the shape of the mock-up fillets is rather symmetric. The results do show that an accurate computation of the centre of mass is essential for grasping success.

As shown in the tests, we find that the path planning fails when the centre of mass is slightly off the true middle, because one of the grippers will then collide with the convex hull of the chicken fillet. To counter this, we would encourage to implement multi-view reconstruction to the pipeline. For test case (I) it was found that for chicken fillets positioned toward the edge of the field of view, the planning success rate was lower as opposed to those more in the middle. This can be explained by looking at the obtained point clouds, as these are only based on one view. Viewing the scene from multiple viewpoints should increase the accuracy of the reconstruction and thus the performance of the pipeline. Multi-view reconstruction was initially implemented, but due to software incompatibilities this could not be tested on the actual robot.

For the grasping, we suggest using different grippers than the Franka finger grippers we used for our tests. These grippers are not optimal for grasping fillets, as they try to grasp a deformable object while having only two small squares as grasping surface. So deformation of the grasped objects is almost certain, possibly resulting in a placement where the fillets are deformed. Furthermore, we compute the convex hull for a static scene, not taking into account possible deforming of the object while the pick-and-place is executed, and thus not guaranteeing collision-avoidance. Even for successful grasps, since the test fillets were not slippery, we expect the grippers to experience difficulty grasping real chicken fillets. Additionally, the maximum width between the grippers makes it difficult to consistently compute the grasp affordance, as was found with the planning failures due to an inaccurate centre of mass. Furthermore, when working with objects that are touching each other, the gripper fingers will likely not be able to grasp them due to being too thick. Therefore, we suggest using thinner grippers with a larger grasping surface, with a larger distance between the gripper fingers than is currently the case.

VIII. Conclusion

In this paper, we presented an automated robotic system for bin-picking of chicken fillets from piles, using deep learning based instance segmentation and 3D object and world reconstruction from depth data. The pipeline consists of perception, point cloud processing, pick-and-place computation, and pick-and-place execution. We validated the instance segmentation on images of real chicken fillets and obtain the average precision (AP). Experiments on the pipeline were done to obtain its planning and execution rate, by testing on mock-up chicken fillets.

We have shown that by training the instance segmentation model on our synthetically rendered images, we can detect real chicken fillet instances in a scene. Additionally, finetuning the segmentation model on a small real dataset was shown to improve the instance segmentation accuracy on real chicken fillets, increasing the mask AP@50:5:95 from 0.68 to 0.78.

Depth data from the RGB-D camera was used to reconstruct a 3D simulated environment for the robot. With this environment in mind, the path planning is computed such that collisions are avoided. The tests on the pipeline with the centre of mass as the point cloud mean showed a planning success rate of 0.87 and a execution success rate of 0.69. Using a centre in the middle of the convex hull showed a planning success rate of 0.73 and an improved execution success rate of 0.81. In some cases, depending on the location of the chicken fillets with respect to the RGB-D camera, we found that the 3D reconstruction of the objects is incomplete, resulting in failed planning or failed grasp execution. We suggest implementing multi-view reconstruction to counter this issue.

The instance segmentation was tested on real chicken fillets, but the grasping only on mock-up objects. The results of the instance segmentation suggest that a path can be planned for real chicken fillets, should they be segmented and reconstructed correctly. As the full pipeline was not deployed on real chicken fillets, but on objects without shiny and slippery surface, it has yet to be shown that the depth data is obtained correctly, and whether the grasping performs similarly. We expect improved grasping results with different grippers.

The way the pipeline is constructed makes it possible to add multiple improvements without needing to make excessive alterations. For instance, the grippers could be exchanged to be more suitable for the objects in question. We already use two different chicken fillet models in the dataset generator, so we could easily add a different object. Then the network could be trained for multiple objects. Additionally, the network could be expanded such that bins can be detected to place the chicken in.

So in conclusion, we showed a promising pipeline for bin-picking of chicken fillets, which can be improved even further.

References

[1] G. A. Nayik, K. Muzaffar, and A. Gull, "Robotics and Food Technology: A Mini Review," *Journal of Nutrition & Food Sciences*, vol. 05, no. 04, 2015, ISSN: 21559600. DOI: 10.4172/2155-9600.1000384.

[2] D. G. Caldwell, S. Davis, R. J. Moreno Masey, and J. O. Gray, "Automation in Food Processing," in *Springer Handbook of Automation*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1041–1059. DOI: 10.1007/978-3-540-78831-7{_}60.

[3] E. Misimi, E. R. Øye, A. Eilertsen, et al., "GRIBBOT – Robotic 3D vision-guided harvesting of chicken fillets," *Computers and Electronics in Agriculture*, vol. 121, pp. 84–100, Feb. 2016, ISSN: 01681699. DOI: 10.1016/j.compag.2015.11.021.

[4] U. Arachchige, S. Chandrasiri, and A. Wijenayake, "Development of automated systems for the implementation of food processing," *Journal of Research Technology & Engineering*, vol. 3, no. 1, pp. 2022–2030, Jan. 2022.

[5] N. Aditama and R. Munir, "Indonesian Street Food Calorie Estimation Using Mask R-CNN and Multiple Linear Regression," in *2022 Second International Conference on Power, Control and Computing Technologies (ICPC2T)*, IEEE, Mar. 2022, pp. 1–6, ISBN: 978-1-6654-5858-0. DOI: 10.1109/ICPC2T53885.2022.9776804.

[6] D. Park, J. Lee, J. Lee, and K. Lee, "Deep Learning based Food Instance Segmentation using Synthetic Data," in *2021 18th International Conference on Ubiquitous Robots (UR)*, IEEE, Jul. 2021, pp. 499–505, ISBN: 978-1-6654-3899-5. DOI: 10.1109/UR52253.2021.9494704.

[7] S. Yarnchalothorn, N. Damrongplasit, S. Chumkamon, and E. Hayashi, "Real-Time Instance Segmentation and Point Cloud Extraction for Japanese Food," *Proceedings of the SICE Annual Conference 2020*, pp. 338–342, Sep. 2020.

[8] A. I. Karoly and P. Galambos, "Automated Dataset Generation with Blender for Deep Learning-based Object Segmentation," in *2022 IEEE 20th Jubilee World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, IEEE, Mar. 2022, pp. 000 329–000 334, ISBN: 978-1-6654-9704-6. DOI: 10.1109/SAMI54271.2022.9780790.

[9] J. Tremblay, A. Prakash, D. Acuna, et al., "Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization," Apr. 2018.

[10] A. Ummadisingu, K. Takahashi, and N. Fukaya, "Cluttered Food Grasping with Adaptive Fingers and Synthetic-Data Trained Object Detection," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, May 2022, pp. 8290–8297, ISBN: 978-1-7281-9681-7. DOI: 10.1109/ICRA46639.2022.9812448.

[11] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," Mar. 2017.

[12] A. Cordeiro, L. F. Rocha, C. Costa, P. Costa, and M. F. Silva, "Bin Picking Approaches Based on Deep Learning Techniques: A State-of-the-Art Survey," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, IEEE, Apr. 2022, pp. 110–117, ISBN: 978-1-6654-8217-2. DOI: 10.1109/ICARSC55462.2022.9784795.

[13] C. Pohl, K. Hitzler, R. Grimm, A. Zea, U. D. Hanebeck, and T. Asfour, "Affordance-Based Grasping and Manipulation in Real World Applications," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2020, pp. 9569–9576, ISBN: 978-1-7281-6212-6. DOI: 10.1109/IROS45743.2020.9341482.

[14] H. Tachikake and W. Watanabe, "A Learning-based Robotic Bin-picking with Flexibly Customizable Grasping Conditions," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2020, pp. 9040–9047, ISBN: 978-1-7281-6212-6. DOI: 10.1109/IROS45743.2020.9340904.

[15] J. Guo, L. Fu, M. Jia, K. Wang, and S. Liu, "Fast and Robust Bin-picking System for Densely Piled Industrial Objects," Dec. 2020.

[16] X. Lou, Y. Yang, and C. Choi, "Collision-Aware Target-Driven Object Grasping in Constrained Environments," Apr. 2021.

[17] T. B. Jørgensen, S. H. N. Jensen, H. Aanæs, N. W. Hansen, and N. Krüger, "An Adaptive Robotic System for Doing Pick and Place Operations with Deformable Objects," *Journal of Intelligent & Robotic Systems*, vol. 94, no. 1, pp. 81–100, Apr. 2019, ISSN: 0921-0296. DOI: 10.1007/s10846-018-0958-6.

[18] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: state of the art," *International Journal of Multimedia Information Retrieval*, vol. 9, no. 3, pp. 171–189, Sep. 2020, ISSN: 2192-6611. DOI: 10.1007/s13735-020-00195-x.

[19] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollar, "Panoptic Segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2019, pp. 9396–9405, ISBN: 978-1-7281-3293-8. DOI: 10.1109/CVPR.2019.00963.

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World," Mar. 2017.

[21] D.-J. Kim, R. Lovelett, and A. Behal, "Eye-in-hand stereo visual servoing of an assistive robot arm in unstructured environments," in *2009 IEEE International Conference on Robotics and Automation*, IEEE, May 2009, pp. 2326–2331, ISBN: 978-1-4244-2788-8. DOI: 10.1109/ROBOT.2009.5152821.

[22] D. Kaljaca, N. Mayer, B. Vroegindeweij, A. Mencarelli, E. v. Henten, and T. Brox, "Automated Boxwood Topiary Trimming with a Robotic Arm and Integrated Stereo Vision," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Nov. 2019, pp. 5542–5549, ISBN: 978-1-7281-4004-9. DOI: 10.1109/IROS40897.2019.8968446.

[23] D. Kaljaca, B. Vroegindeweij, and E. Henten, "Coverage trajectory planning for a bush trimming robot arm," *Journal of Field Robotics*, vol. 37, no. 2, pp. 283–308, Mar. 2020, ISSN: 1556-4959. DOI: 10.1002/rob.21917.

[24] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," Nov. 2017.

[25] F.-J. Chu, R. Xu, and P. A. Vela, "Real-World Multiobject, Multigrasp Detection," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3355–3362, Oct. 2018, ISSN: 2377-3766. DOI: 10.1109/LRA.2018.2852777.

[26] M. Sun and Y. Gao, "GATER: Learning Grasp-Action-Target Embeddings and Relations for Task-Specific Grasping," *IEEE Robotics and Automation Letters*, vol. 7, no. 1, pp. 618–625, Jan. 2022, ISSN: 2377-3766. DOI: 10.1109/LRA.2021.3131378.

[27] B. Calli, M. Wisse, and P. Jonker, "Grasping of unknown objects via curvature maximization using active vision," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Sep. 2011, pp. 995–1001, ISBN: 978-1-61284-456-5. DOI: 10.1109/IROS.2011.6094686.

[28] D. Kanoulas, J. Lee, D. G. Caldwell, and N. G. Tsagarakis, "Visual Grasp Affordance Localization in Point Clouds Using Curved Contact Patches," *International Journal of Humanoid Robotics*, vol. 14, no. 01, p. 1 650 028, Mar. 2017, ISSN: 0219-8436. DOI: 10.1142/S0219843616500286.

[29] M. Gorner, R. Haschke, H. Ritter, and J. Zhang, "MoveIt! Task Constructor for Task-Level Motion Planning," in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, May 2019, pp. 190–196, ISBN: 978-1-5386-6027-0. DOI: 10.1109/ICRA.2019.8793898.

[30] D. Horvath, G. Erdos, Z. Istenes, T. Horvath, and S. Foldi, "Object Detection Using Sim2Real Domain Randomization for

Robotic Applications," *IEEE Transactions on Robotics*, pp. 1–19, 2022, ISSN: 1552-3098. DOI: 10.1109/TRO.2022.3207619.

[31] Y. Li, S. Xie, X. Chen, P. Dollar, K. He, and R. Girshick, "Benchmarking Detection Transfer Learning with Vision Transformers," Nov. 2021.

[32] T.-Y. Lin, M. Maire, S. Belongie, et al., "Microsoft COCO: Common Objects in Context," May 2014.

[33] K. Wada, *Labelme: Image Polygonal Annotation with Python*, 2022. [Online]. Available: https://github.com/wkentaro/labelme.

[34] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A Survey on Performance Metrics for Object-Detection Algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, IEEE, Jul. 2020, pp. 237–242, ISBN: 978-1-7281-7539-3. DOI: 10.1109/IWSSIP48289.2020.9145130.

[35] Franka Emika, *ROS integration for Franka Emika research robots*. [Online]. Available: https://github.com/frankaemika/franka_ros.

[36] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A Modern Library for 3D Data Processing," Jan. 2018.

[37] S. Chitta, "MoveIt!: An Introduction," in *Robot Operating System (ROS) The Complete Reference (Volume 1)*, A. Koubaa, Ed., vol. 625, Switzerland: Springer International Publishing Switzerland, 2016, ch. 3, pp. 3–27, ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9.