Grasping and Sorting Cutlery in an Unconstrained Environment with a 6 DoF Robotic Arm and an RGB+D Camera

Ricardo Vermelho, Luís A. Alexandre
NOVA LINCS and Departamento Informática
Universidade da Beira Interior, Covilhã, Portugal
{ricardo.vermelho, luis.alexandre}@ubi.pt

***Abstract*—Object manipulation is the ability to perform tasks of grabbing, pushing or moving objects. Although this is something humans do naturally and without realizing it, for robots it is still hard to do. Despite that, robotic manipulation has seen an exponential evolution over the years, although there are still many open challenges in the field. One of these challenges is cutlery manipulation since this kind of object has features that make them hard to manipulate, like the long and thin shape, the metallic specular reflections and the fact that cutlery is usually mixed in a container. In this paper, we propose a method that sorts cutlery, which includes three main steps: the detection of the particular object in the point cloud, the subsequent grasping that can be helped by pushing motions powered by an algorithm that was previously developed, and finally the placing of the object in a particular container, according to its type. We show that we are able to perform this sorting task in simulation using a UR3 robotic arm and a RGB+D camera. We explore different settings and improve a previous pushing and grasping method.**

***Index Terms*—Robotic Manipulation, Deep Neural Networks, Object Detection, Object Grasping.**

I. INTRODUCTION

Industrial and technological evolution brought robots into our lives. They are present in factories and are beginning to appear in our houses. However, according to [1] it is still hard for robots to perform household chores. They evaluated a PR2 robot that had to make breakfast and clean the table. The robot took 90 minutes to complete the task, showing that it was feasible but very slow and prone to errors.
Manipulation is a very important capability if one wishes to have robots performing everyday tasks. Much work has been done in the area [2], [3], [4], but there are still many open challenges. One of these is for the robots to be able to grasp cutlery, since these objects have several features that make them hard to grasp: they are thin and long, they are shiny since they are usually metallic (this makes it difficult to visually detect them prior to the actual grasping actions) and they are usually lying on a flat surface or mixed together in a container. In this paper we focus on overcoming some of these problems and developing a system that is not just able to grasp cutlery but also to sort it into containers. We build upon a previous work [5], called Visual Pushing and Grasping (VPG), and combine it with YOLO [6] to be able to detect and recognize the different cutlery objects and their positions in the scene, and using both pushing and grasping, enables the system to sort cutlery into different containers, starting from a single container. Our contributions are the following:

1) Explore the impact of using different types of networks, such as DenseNet169, MobileNetV2, MobileNetV3-Large, VGG19 and VGG19-Batch, and different number of batch-normalization layers for the VPG;
2) Evaluating different number of rotations to the RGB+D image taken from the simulation before feeding these new rotated images to the neural networks;
3) Investigate the impact of the convolution kernel size and biased neural networks, on the system performance;
4) We also propose two new metrics for evaluating the sorting tasks.

II. RELATED WORK

A. Murali *et al* [4] state that grasping objects is still an hard challenge, and even more if performed in cluttered environments, thus they propose a method that plans grasps for 6-DOF robots from partial point clouds. Their method takes an RGB+D image as input, from which they select the target object, compute the cropped 3D point cloud and the respective grasping points and obtained 80.3% of grasp success. Peiyuan Ni *et al* [7] demonstrated two situations where it can be difficult to grasp objects, namely when the objects are inside a container or box and are at the corner or edge of the box, which makes the grasping action much harder to perform with success. The proposed method aims to increase grasp access by combining a forgetting mechanism in the grasp's quality function to uphold a pushing action. They trained the algorithm in simulation environment with YCB dataset objects [8] and also performed real-world experiments, obtaining 86.67% of completion using YCB objects and 83.37% of completion using novel objects, in both environments. Andy Zeng *et al* [5] showed that robotic arms can grasp objects based on location when supported by a reinforcement learning algorithm and a depth camera, without the need to be supervised and also showed that sometimes performing a push action benefits the upcoming grasping action for certain objects. Their method is

This work was supported by NOVA LINCS (UIDB/04516/2020) with the financial support of FCT – Fundação para a Ciência e a Tecnologia, through national funds, and by project 026653 (POCI-01- 0247-FEDER-026653) INDTECH 4.0 – New technologies for smart manufacturing, cofinanced by the Portugal 2020 Program (PT 2020), Compete 2020 Program and the European Union through the European Regional Development Fund (ERDF).

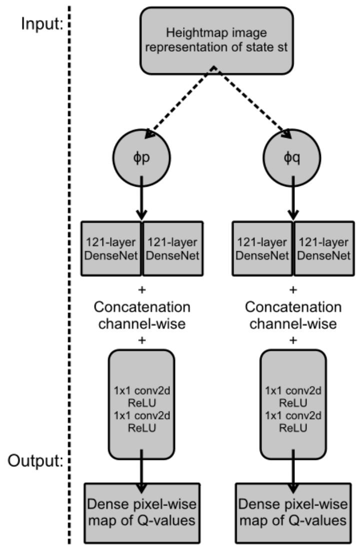

Input:
Heightmap image representation of state st
ϕp
ϕq
121-layer DenseNet
121-layer DenseNet
121-layer DenseNet
121-layer DenseNet
+ Concatenation channel-wise +
+ Concatenation channel-wise +
1x1 conv2d ReLU 1x1 conv2d ReLU
1x1 conv2d ReLU 1x1 conv2d ReLU
Output:
Dense pixel-wise map of Q-values
Dense pixel-wise map of Q-values

Fig. 1. Architecture of the VPG algorithm based on the description in [5].

called VPG, (see Fig. 1) and combines two identical structures, each of them with a convolutional neural network, pre-trained Densenet121 [9], to process 16 different rotations of the original RGB+D image and produce in total 32 Q-value maps containing the locations where applying one of the possible actions is better. In the end, only the best Q-value map is applied and the action can be either a push or a grasp. For real-world cluttered environments, this method obtained 83.30% of grasp success and in cluttered simulation environments, it obtained 77.20% of grasp success rate. Our work differentiates from the aforementioned, as our focus is to detect and grasp cutlery objects by type. We use YOLO version 5X for object detection and adapt VPG [5] to grasp the targeted object and deliver it to the desired drop point. We evaluate our method in terms of grasp success and the ratio of successful and well placed objects.

III. METHOD

Our method is illustrated in Fig. 2. We adapt [5], which already performs push and grasp actions in objects on a computed location based on Q-value maps that indicate how good performing one of the actions is, in a certain location. One contribution is the introduction of a module that contains an object detection model, that will work together with VPG to detect the type of objects present in the table and tell the robot to grasp a specific object and drop it in a specific container. We also explore different types of networks and layers for VPG.

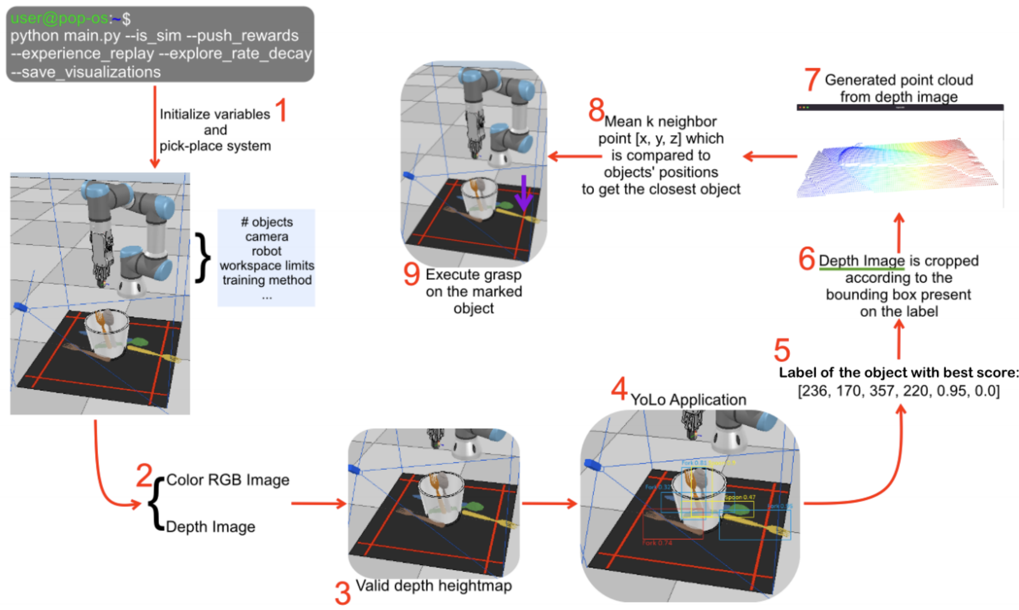
The pipeline of our method starts with the initialization of the simulation's pick-place system, followed by a routine that repeats in every epoch of the simulation (points 2 to 9). After the objects are all placed in the simulation table, we take an RGB image and a depth image (2nd point) to be able to create a valid depth heightmap, that contains values for both RGB and depth channels, in the 3rd point. In point 4, we apply YOLO to the RGB image to obtain the labels of the objects in simulation. These labels are composed of 6 values, the first 4 numbers indicate the starting and ending point of the box surrounding each object in the image, the 5th number indicates the level of confidence attributed by the object detection model to an object as it being of a certain class and lastly, the last number corresponds to the class to which the object belongs. Then, we select the object with the label with best score and crop the depth image in the position where the object is (points 5 and 6). As points 7 and 8 show, in Fig. 2, we convert the cropped depth image to a point cloud of the object and apply the k-means algorithm to obtain the coordinates of the object furthest from the robot, because since the camera is in front of the robot, the points that are further away from the camera are closer to the robot. The conversion and application of the k-means algorithm is explained as follows:

1) Retrieve the depth image of the current epoch/iteration and crop it according to the bounding box of the object with better confidence value attributed by YoLo;
2) Initialize the camera intrinsic values - [[f 0 w/2], [0 f h/2], [0 0 1]], where f is the focal length, which in our case is 618.62. w/2 and h/2 represent the central point of the image;
3) Create point cloud from depth image, with depth_scale=10000 and depth_trunc=10000. This way, all the values in the point cloud are calculated in meters;
4) Initialize camera pose using camera position, camera orientation and camera rotation matrix, i.e., simulation world position [x, y, z], its orientation angles with each axis and the rotation matrix given as a list of values in Euler angles;
5) Convert point cloud from camera coordinate system to robot coordinate system;
6) Apply zFar=10 and zNear=0.01 transformation to all points in the point cloud;
7) Sort surface points of point cloud by z value;
8) Collect the point with largest z from the point cloud;
9) Apply k-means algorithm to obtain the average point from point cloud;
10) Get average coordinates for the cluster of points obtained from k-means.

Afterwards, we compare to the coordinates of the objects given by one of the methods present in VPG algorithm, and the object with coordinates closer in (x,y,z) to those obtained from the steps 7 and 8, will be the object selected to be grasped or pushed.

IV. EVALUATION METRICS

In section V and VI, we talk about the experiments made with the baseline method VPG [5] and the experiments made with our contribution with the addition of the object detection model to this pushing and grasping algorithm. We assess



```
user@pop-os:~$
python main.py --is_sim --push_rewards
--experience_replay --explore_rate_decay
--save_visualizations
```

Initialize variables and pick-place system 1
# objects
camera
robot
workspace limits
training method
...
2 Color RGB Image
Depth Image
3 Valid depth heightmap
4 YoLo Application
5 Label of the object with best score: [236, 170, 357, 220, 0.95, 0.0]
6 Depth Image is cropped according to the bounding box present on the label
7 Generated point cloud from depth image
8 Mean k neighbor point [x, y, z] which is compared to objects' positions to get the closest object
9 Execute grasp on the marked object

Fig. 2. Pipeline of our method: initialize pick-place system (1), obtain RGB and depth images (2), create a depth heightmap containing depth values (3), obtain labels of the objects with YOLO (4), get object with the label with best score and use it to crop the depth image (5 and 6), generate a point cloud and obtain an average point from applying the k-means algorithm (7 and 8) and execute action on an object (9).

the whole simulations, for both VPG and our versions of the contribution, in terms of push and grasp ratios. But our contribution is also evaluated in terms of sorting accuracy. The first two metrics used are related to push and grasp ratios and the last two metrics evaluate our contribution for its sorting accuracy, since we want to know how well the detection, grasping and placement of the objects went:

- M_{grasps} is the average of the ratio between number of successful grasps g_i, and all made grasps ng_i, of each simulation epoch i, given by: $M_{grasps} = \frac{1}{N_g}\sum_{i=1}^{N_g}\frac{g_i}{ng_i}$ where N_g corresponds to the total number of grasps made during one experiment/simulation.
- M_{pushes} is the average of the ratio between number of successful pushes p_i, and all made pushes np_i, of each simulation epoch i, given by: $M_{pushes} = \frac{1}{N_p}\sum_{i=1}^{N_p}\frac{p_i}{np_i}$, where N_p corresponds to the total number of pushes made during one experiment/simulation.
- The ratio of successful and well placed grasps in the whole universe of all successful grasps (well placed or not):

$$ACC_{grasps1} = \frac{\text{\# successful and well placed grasps}}{\text{total \# successful grasps}};$$

- The ratio of successful and well placed grasps in the whole universe of all grasps (successful or not):

$$ACC_{grasps2} = \frac{\text{\# successful and well placed grasps}}{N_g}.$$

V. EXPERIMENTAL SETUP

We evaluate the baseline method for object grasping [5] and all our proposed modifications in various simulation experiments. The simulation environment used was CoppeliaSim v4.1, with an UR3 robotic arm, 3 containers for sorted objects and also 3 to 6 objects, in order to test non-cluttered and cluttered environments. The experiments are made on the Ubuntu operating system, using an AMD Ryzen7 2700X CPU, 32GB DDR4 RAM and Nvidia Geforce RTX 3060 12GB, and since we are running an algorithm for pushing and grasping and an object detection model, it takes up to 10GB of dedicated video memory for computational purposes.

Our experiments are focused on checking if introducing some changes in the two identical structures of the VPG algorithm would produce better results and also improving the baseline algorithm by adding an object detection model to detect the objects available to grasp and facilitate the object dropping decision in the right container by knowing its type.

A. *Improving the Baseline Method*

The first proposed changes involve trying different neural networks and changing other configurations regarding the networks to improve the VPG. They contemplate the following list:

- Number of rotations applied to the RGB+D image taken from the simulation;
- Make use of other 5 different neural networks, such as: DenseNet169, MobileNetV2, MobileNetV3-Large, VGG19 and VGG19-Batch;
- Different number of batch-normalization layers;

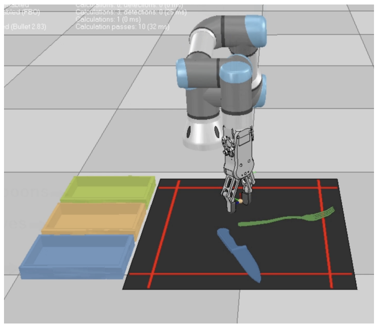


Fig. 3. Experimental setup used by our contribution. In the image: Universal Robot 3, cutlery objects from TurboSquid [10] and the 3 containers for the sorted cutlery (left).

- Increased size for the kernel of the convolutions;
- And, using biased neural networks.

These experiments are performed using block shaped objects that were used in the original VPG paper. Note that these are much easier to grasp than the cutlery objects used in the experiments performed afterwords. The experiments are all divided by the category of number of rotations applied to the RGB+D image taken from the simulation and inside that category is where we apply other changes to the method. An image of the experimental setup similar to the one used by VPG can be seen in Fig. 3. The first experiments made reflected on the first 3 bar charts used block-shaped objects for the simulations. The introduced changes are as follows:

1) 2 Batch-Normalization layers;
2) 2 Batch-Normalization layers with Bias;
3) 2 Batch-Normalization layers with Convolutional Kernel size of 2;
4) 3 Batch-Normalization layers;
5) 4 Batch-Normalization layers.

Thus, the initial tests were made with the application of 16 rotations on the original RGB+D image taken from the simulation.

Under the category of the 16 rotations, we changed VPG according to the number of batch-normalization layers used. Fig. 4 contains the results for the experiments made. We may see 5 different values in the x axis, and that is because we used different number of batch-normalization layers as sub-category, inside each one of those, we performed 10 simulations for each one of the neural networks present in the bar chart's legend and the % in the chart is the average of those 10 simulations.

The second batch of experiments made have as category, 8 rotations of the RGB+D image and the number of Batch-Normalization layers used are respectively 2 and 3, unbiased and with no increase of Convolutional Kernel size. For these tests we also simulated 10 times for each of the neural networks and calculated the average value. The results are presented in Fig. 5.

The last experiments made using block-shaped objects con-

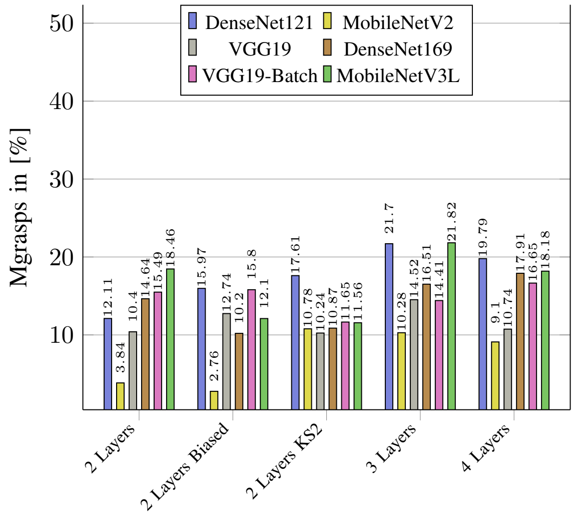


Fig. 4. VPG 16 rotations RGB+D tests. VPG using a 3 batch normalization layer configuration obtained the best results of all experiments when using pre-trained DenseNet121 and pre-trained MobileNetV3-Large.

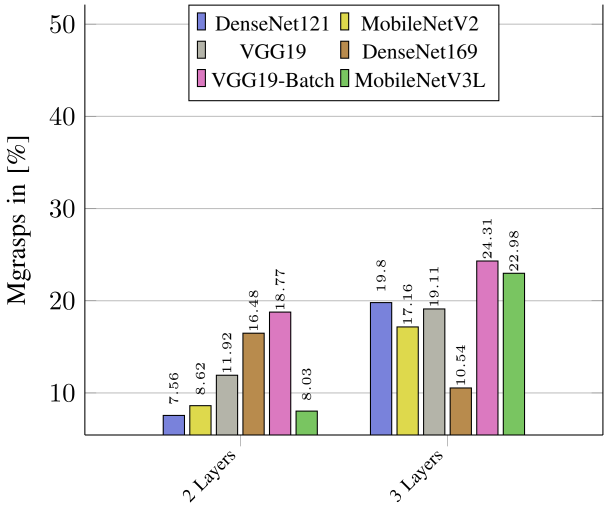


Fig. 5. VPG 8 rotations RGB+D tests. VPG using a 3 batch normalization layer configuration obtained the best results of all experiments in this category when using pre-trained VGG19-Batch and pre-trained MobileNetV3-Large.

template 32 rotations of the original RGB+D image and the number of Batch-Normalization layers used are respectively 2 and 3, unbiased and with no increase of the convolutional kernel size. Once again, for these tests we simulated 10 times for each of the neural networks and calculated the average value. These results appear in Fig. 6.

B. *Testing with Cutlery Objects*

Following the line of adjusting the VPG method to cutlery objects, we used models taken from YCB Dataset [8] to perform more experiments, now using cutlery.

Once more, we made experiments based on the number of rotations applied to the original RGB+D image taken from simulation. These rotations can be either 8 or 16 or 32.

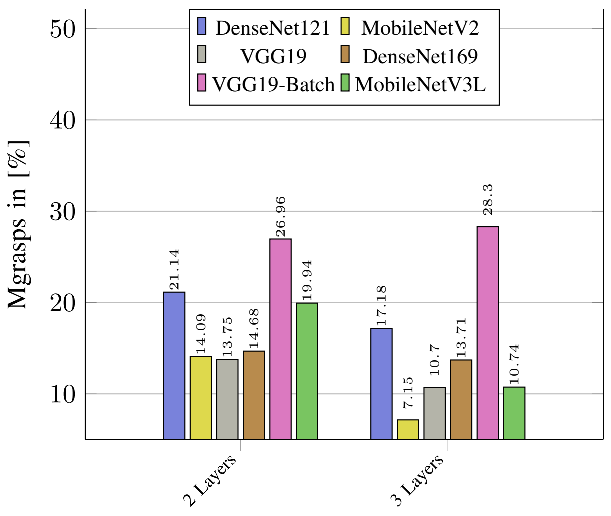


Fig. 6. VPG 32 Rotations RGB+D Tests. In these experiments the best results were obtained using pre-trained MobileNetV3-Large for both batch normalization configurations.

TABLE I
RESULTS FOR 8/16/32 ROTATIONS OF VPG USING YCB DATASET MODELS [8].

			Rotations		
			8	16	32
			Mgrasps [%]		
Batch Norm Layers	**2**	**DenseNet121**	Not tested	Not tested	**26.29%**
		MobieNetV3L	Not tested	20.99%	18.79%
		VGG19-Batch	Not tested	Not tested	23.39%
	3	**DenseNet121**	Not tested	21.75%	Not tested
		MobieNetV3L	**26.08%**	**26.96%**	Not tested
		VGG19-Batch	17.34%	Not tested	21.99%
	4	**DenseNet121**	Not tested	17.81%	Not tested
		MobieNetV3L	Not tested	24.92%	Not tested

However, the number of Batch-Normalization layers applied differs, i.e., the number of layers used for 8 rotations is different from those used with 16 rotations and for 32 rotations. Also, we decided to test the baseline method with the neural networks that showed better results in previous simulations with the block-shaped objects, so Table I contains some cells that say ”Not tested”, meaning that configuration wasn't tested due to previous poor results.

Figures 4, 5 and 6 and Table I have the results for the metric Mgrasps and not for Mpushes, because the original method validates a push action even if it was performed in a location where there weren't any objects, causing the Mpushes metric to always be 100%, which is not true.

Lastly, and because it has more to do with what we propose, we ran experiments using a combination of the baseline method VPG [5] and the object detection model YoLo [6]. We made two versions of our proposal, where the difference between them is the way they pre-process the image taken from simulation with the object detection model. We call them OursV1 and OursV2. In Oursv1 it is purely based on the object closest to the robot (distance to it) and it also makes pushing actions when a grasp failed. However in Oursv2, besides looking for the object closest to the robot, we also check if said object has other objects close to it and therefore avoiding making a grasp when it could possibly produce a different outcome than what we were expecting, thus applying a push action instead. The experiments ran for 63 and 250 epochs, using cutlery object models from YCB dataset [8] and TurboSquid [10] without using a container for the objects that are waiting to be grasped; different number of objects per experiment, either 3 or 6 objects; the original DenseNet121 configuration; the 16 rotations applied to the RGB+D image and both Mgrasps and Mpushes metrics were evaluated. It is also important to say that, both datasets were used initially with VPG algorithm to obtain images to use in the fine tuning process of the object detection model.

TABLE II
RESULTS FOR 63 EPOCHS OF OUR MODEL USING 3 AND 6 CUTLERY OBJECTS.

		Mgrasps [%]		Mpushes [%]	
		OursV1	**OursV2**	**OursV1**	**OursV2**
TurboSquid # Objects	3	23.49	**24.24**	79.78	**95.24**
	6	18.24	**34.32**	**86.43**	63.49
YCB # Objects	3	22.81	**33.58**	**92.70**	0.0
	6	**22.69**	7.88	68.27	**80.95**

TABLE III
RESULTS FOR 250 EPOCHS OF OUR MODEL USING 3 AND 6 CUTLERY OBJECTS.

		Mgrasps [%]		Mpushes [%]	
		OursV1	**OursV2**	**OursV1**	**OursV2**
TurboSquid # Objects	3	22.14	**32.14**	**98.59**	87.86
	6	19.71	**24.46**	85.65	**96.27**
YCB # Objects	3	13.35	**26.21**	**80.87**	68.09
	6	**25.77**	27.42	**91.70**	66.72

VI. DISCUSSION

From all the experiments made with the baseline method using block-shaped objects, we see that the original configuration using the neural network DenseNet121 isn't always the one that has the best results, in fact we can improve the results of the VPG using neural networks such as MobileNetV3-Large and VGG19-Batch.

For example, when making use of the VPG with 16 rotations of the original RGB+D image (see Fig. 4), we got 21.70% and 21.82% of average grasp success ratio Mgrasps, for DenseNet121 and MobileNetV3-Large with a 3 batch-normalization layer, respectively. However, when testing the baseline method with 8 rotations of the original RGB+D image (see Fig. 5), we see that MobileNetV3-Large and VGG19-Batch had the best results for a 3 batch-normalization layer configuration with 22.98% and 24.31% of Mgrasps respectively. Meanwhile, using VPG with a 32 rotation of the original RGB+D image configuration (see Fig. 6) proved to be interesting, with MobileNetV3-Large having the best Mgrasps results for both 2 and 3 batch-normalization layer configurations, 26.96% and 28.30% respectively.

TABLE IV
$ACC_{grasps1}$ [%] RESULTS FOR OUR METHOD USING TURBOSQUID AND YCB MODELS.

		ACC$_{grasps1}$[%]			
		OursV1		**OursV2**	
		63 epochs	250 epochs	63 epochs	250 epochs
TurboSquid	3	1/8 = 12.50%	1/34 = 2.94%	**12/21 = 57.10%**	**18/65 = 27.69%**
	6	**2/5 = 40.00%**	**3/32 = 9.40%**	5/17 = 29.40%	2/34 = 5.88%
YCB	3	4/9 = 44.44%	**3/22 = 13.63%**	3/17 = 17.60%	7/45 = 15.55%
	6	**3/5 = 60.00%**	3/36 = 8.33%	**2/8 = 25.00%**	**3/13 = 23.07%**

TABLE V
$ACC_{grasps2}$ [%] RESULTS FOR OUR METHOD USING TURBOSQUID AND YCB MODELS.

		ACC$_{grasps2}$[%]			
		OursV1		**OursV2**	
		63 epochs	250 epochs	63 epochs	250 epochs
TurboSquid	3	1/36 = 2.77%	1/142 = 0.70%	**12/62 = 19.40%**	**18/227 = 7.92%**
	6	**2/34 = 5.88%**	**3/141 = 2.12%**	5/60 = 8.33%	2/155 = 1.29%
YCB	3	**4/36 = 11.11%**	**3/136 = 2.20%**	**3/60 = 5.00%**	7/156 = 4.48%
	6	3/34 = 8.82%	3/143 = 2.09%	2/57 = 3.50%	**3/39 = 7.69%**

Nevertheless, using VPG with cutlery objects for the previous best configurations (see Table I), showed that using a:

- 2 layer batch-normalization had the best Mgrasps result with DenseNet121 and 32 rotations with 26.29%;
- 3 layer batch-normalization had best results for Mgrasps using 8 and 16 rotations with MobileNetV3-Large, with 26.08% and 26.96% respectively.

In contrast (see Table II), our contribution compared to VPG using YCB Dataset [8] objects, had the best result for Mgrasps - 33.58%, when using only 3 cutlery objects and for a short number of epochs. We tested the two versions of our method with other 3D cutlery models from TurboSquid and their best result was also obtained when using a short number of epochs, but with a more cluttered scenery (6 objects) - 34.32% and with OursV2. There is room for improvement, however we can think of one reason why Mgrasps percentage was low and it contemplates the fact that these kind of objects resemble the metallic look of the real-world objects which makes them harder for the object detection model to detect and also, because of their thin structure which makes them harder to grasp, even in simulation conditions.

We were able to measure Mpushes for all experiments with our contribution and the best result – 98.59% was achieved with OursV1 (see Table III), when using 3 TurboSquid cutlery object models and for 250 epochs.

As Tables IV and V show, the ratio of successful and well placed grasps in the universe of all successful grasps ($ACC_{grasps1}$) had its best result with OursV1 – 60.00%, using 6 YCB object models, a small number of simulation epochs (63) and the ratio of successful and well placed grasps in the universe of all made grasps ($ACC_{grasps2}$) had its best result with OursV2 – 19.40%, using 3 TurboSquid object models and again, during a small number of simulation epochs (63).

VII. CONCLUSION

In this paper, we propose a method that contributes to the robotic manipulation field, particularly in robotic manipulation of cutlery, by combining a pushing and grasping algorithm for robotic manipulation with an object detection model to be able to detect, grasp and separate different types of cutlery in different containers. By being able to detected the objects and their type, i.e., if they are forks or knives or spoons, we are able to apply pushing or grasping actions on them and tell the robot to place them in the respective container according to their type. We have also shown that there are convolutional neural networks that have better performance and obtain improved results than the original pre-trained DenseNet121 used in VPG, such as the MobileNetV3-Large and VGG19-Batch. Finally, we also explored the impact of the usage of different kernel sizes, batch normalization layers and biased networks in the performance of the VPG as an attempt to improve the overall results and can conclude that amongst all the different configurations tested, the three batch normalization layer configuration produced the best results. We make our code available at: github.com/RicardoVermelho/VPG-YOLO-cutlery.

REFERENCES

[1] G. Kazhoyan, S. Stelter, F. K. Kenfack, S. Koralewski, and M. Beetz, "The robot household marathon experiment." [Online]. Available: https://arxiv.org/abs/2011.09792/

[2] H. Nambiappan, K. Kodur, M. Kyrarini, F. Makedon, and N. Gans, "Mina: A multitasking intelligent nurse aid robot," *The 14th PErvasive Technologies Related to Assistive Environments Conference*, 2021.

[3] R. López-Sastre, M. Baptista-Ríos, F. J. Acevedo-Rodríguez, S. P. da Costa, S. Maldonado-Bascón, and S. Lafuente-Arroyo, "A low-cost assistive robot for children with neurodevelopmental disorders to aid in daily living activities," *International Journal of Environmental Research and Public Health*, vol. 18, 2021.

[4] A. Murali, A. Mousavian, C. Eppner, C. Paxton, and D. Fox, "6-dof grasping for target-driven object manipulation in clutter," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6232–6238, 2020.

[5] A. Zeng, S. Song, S. Welker, J. Lee, A. Rodriguez, and T. Funkhouser, "Learning synergies between pushing and grasping with self-supervised deep reinforcement learning," 2018.

[6] Ultralytics, "Pytorch yolov5." 2020. [Online]. Available: http://dx.doi.org/10.5281/zenodo.3908559

[7] P. Ni, W. Zhang, H. Zhang, and Q. Cao, "Learning efficient push and grasp policy in a totebox from simulation," *Advanced Robotics*, vol. 34, pp. 873 – 887, 2020.

[8] B. Calli, A. Singh, J. Bruce, A. Walsman, K. Konolige, S. Srinivasa, P. Abbeel, and A. Dollar, "Yale-cmu-berkeley dataset for robotic manipulation research." *The International Journal of Robotics Research*, vol. 36, p. 027836491770071, 04 2017. [Online]. Available: http://dx.doi.org/10.1177/0278364917700714

[9] "Densenet121 pytorch implementation." [Online]. Available: https://pytorch.org/hub/pytorch_vision_densenet/

[10] "Turbosquid: Online 3d models website." [Online]. Available: https://www.turbosquid.com/