Review on GGCNN: A Real-time, Generative, Grasp Synthesis Approach

Wang Siyuan, Zhao Ruihan, Su Zhaowen, Liu Ziyu, Lin Peijun

***Abstract*—The development of intelligent robotic grasping (IRG) is important in various household and industrial applications, but it is complex due to the different shapes of objects and environmental factors. Deep convolutional neural networks, such as the generative grasping CNN (GG-CNN), have facilitated IRG by generating grasping strategies for unknown objects. However, factors such as dataset and annotation methods, type of robotic arm and simulation environment, and training methods and hardware devices impact the accuracy and applicability of the model. In this study, we reproduced GG-CNN and proposed some improvements after identifying potential limitations. Under our task setting, GGCNN achieved an average of 83.9% successful grasp rate and a 101.12ms prediction duration, while GGCNN2 achieved 93.7% and 116.8ms on average. Based on our experimental results, GGCNN and GGCNN2 can effectively predict planar grasping poses from a depth image at a fast rate. GGCNN has relatively lower computational requirements, while GGCNN2 requires a little more computing time and shows a better performance. However, our work reveals that GGCNN's grasp depth prediction is unreliable for objects with hollow structures.**

I. Introduction

Intelligent robotic grasping (IRG) is a challenging problem in various household and industrial applications. Precisely and intelligently grasping unfamiliar objects in unknown environments is complex because of the different shapes of the objects and the impact of environmental factors. The emergence of deep convolutional neural networks (CNNs) and new designs of various CNN architectures have facilitated the development of IRGs, allowing for the planning of grasping strategies for unknown objects.

The deep learning grasping approaches have been actively investigated recently. Peter Corke and Morrison provide a way to predict multiple grasps and calculates the grasp scores based on FCNs. By using feature maps representing pixelwise grasps on the images, the proposed generative grasping CNN (GG-CNN) in predicts a set of grasps with corresponding quality from the depth image. The obvious advantage is that dense grasp generation and the corresponding primary evaluation can be accomplished with only a single prediction of the FCN.

However, several factors, such as the dataset and annotation methods employed, the type of robotic arm and simulation environment used, as well as the training methods and hardware devices utilized, have a significant impact on the accuracy and applicability of the model. For this reason, we intend to reproduce Peter Corke's work (GG-CNN)[9] and analyze the performance differences between our approaches. And finally, we have identified some potential limitations of GG-CNN and have proposed some improvement strategies.

II. Related Work

Grasp synthesis is the process of developing a stable robot grasp for a specific object. This topic has been widely studied, and numerous techniques have been proposed. These approaches can be generally classified into analytic and empirical methods (Bohg et al., 2014)[1]. Analytic methods use mathematical and physical models to determine stable grasps, but they do not perform well in the real world due to challenges in modeling physical interactions between the manipulator and the object (Rubert et al., 2017)[11].

Empirical methods, on the other hand, rely on experience-based approaches and models to determine grasping points. Some of these methods use offline databases of known objects or object classes, but they are unable to generalize to new objects. In the 2017 Amazon Robotics Challenge, many teams were presented with novel objects just before their challenge run to test their ability to generalize (Morrison et al., 2018b)[10].

Deep-learning techniques have improved object grasping in recent years, with numerous techniques developed by several researchers (Lenz et al., 2015)[4]. Typically, these approaches sample grasp candidates from an image or point cloud and rank them using convolutional neural networks before executing the best grasp candidate open-loop. However, this method demands precise calibration between camera and robot, robot control, a static environment, and can be time-consuming.

To overcome the issue of long execution times, various techniques are used to reduce the number of grasp candidates. For instance, some approaches pre-process and eliminate unfeasible grasp candidates, while others simultaneously predict the quality of a specific set of grasp candidates (Johns et al., 2016)[3]. However, these approaches trade off execution time against the number of grasp candidates that are sampled. As a result, they may overlook some potential grasps.

Instead of sampling grasp candidates, Douglas Morrison, Peter Corke, and Jurgen Leitner et al. present a novel approach to perform object-independent grasp synthesis from depth images via deep neural networks. Their generative grasping convolutional neural network (GG-CNN) overcomes shortcomings in existing techniques, namely discrete sampling of grasp candidates and long computation times while achieving better performance.

We reproduce their work, address the issues of execution time and grasp sampling by directly generating grasp poses for every pixel simultaneously, using a comparatively small neural network.

III. Data

A. Datasets

In order to train the GG-CNN to perform dense, pixel-wise grasp prediction we require a densely labeled grasping dataset. First, we used the smaller, hand-labeled Cornell Grasping Dataset (Lenz et al., 2015)[4]. It represents antipodal grasps in RGB and depth images as rectangles using pixel coordinates (Yun Jiang et al., 2011)[2].

The Cornell Grasping Dataset contains 885 RGB-D images of real objects, with 5,110 human-labeled positive and 2,909 negative grasps. While this is a relatively small grasping dataset compared with some other, synthetic datasets (Mahler et al., 2017, 2016)[7], the data best suits our pix-elwise grasp representation as multiple labeled grasps are provided per image. This is a more realistic estimate of the full pixel-wise grasp map, than using a single image to represent one grasp, such as in Mahler et al. (2017)[6].

B. Simulation Environment

Due to the limitations of real-world devices and the convenience of developing simulation environment packages, we chose to use pybullet to build our simulation environment. In the field of machine learning, PyBullet is commonly used for reinforcement learning research and development. PyBullet provides a physics simulation environment that allows us to create complex environments for training and testing reinforcement learning agents. It allows for simulation of complex dynamics environments, including contact dynamics, deformable objects, and soft-body dynamics. Another advantages of PyBullet is it provides a flexible and intuitive way to interact with the simulation environment, which is useful for debugging and visualization of machine learning algorithms.

We deployed the Panda robotic arm in the pybullet simulation environment. The Panda robotic arm is a collaborative robot developed by the German robotics company Franka Emika. It consists of seven interconnected joints, allowing it to move with six degrees of freedom. The Panda robotic arm is designed to be lightweight, precise, and easy to use, making it suitable for a wide range of industrial and research applications, such as pick-and-place operations and material handling, and provides a safe, efficient, and cost-effective way to test and develop our generative grasping convolutional neural network (GG-CNN) algorithms.

IV. Methods

A. Data Processing

In the general two-finger manipulator plane grasping model, the vertical desktop grasping of a mechanical arm is restricted, meaning that the roll and pitch of the arm are fixed values. The model predicts the grasping parameters [x, y, z, ϕ, w], where [x, y, z] represents the world coordinates of the grasping point. ϕ represents the rotation angle of the manipulator along the z-axis or the angle between the closing direction of the manipulator and the horizontal axis of the image. The value

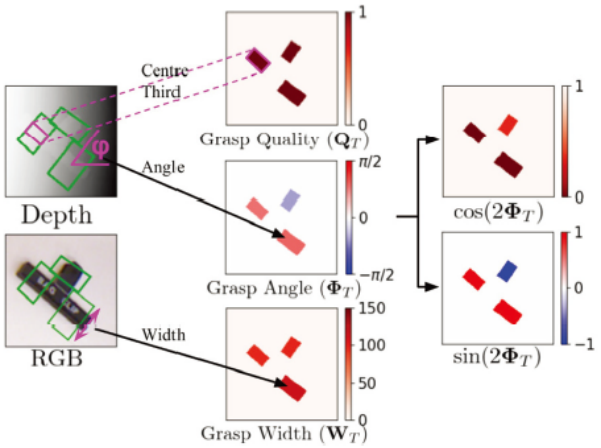

Fig. 1. Generation of training data for our GG-CNN. Left: The cropped and rotated depth images from an antipodal grasping dataset, with the ground-truth positive grasp rectangles shown in green. NB: The RGB image is added for illustration only and is not used by our system. Right: The grasp quality ($\tilde{\mathbf{Q}}$), grasp angle ($\tilde{\phi}$), and grasp width ($\tilde{\mathbf{W}}$) images to train our network. The angle is further decomposed into cos ($2\tilde{\phi}$) and sin ($2\tilde{\phi}$) for training.

of z can be either output by the neural network or calculated using a separate algorithm.

When it comes to grabbing a rectangular box, the algorithm needs to predict the opening width of the manipulator, denoted as 'n', and the size of the manipulator claw or grip set, denoted as 'm'. The product of 'n' and 'm' represents the grasp of the rectangular box. In the Cornell dataset, objects are labeled with rectangular bounding boxes representing the grasp region. To conform to the input requirements of the GGCNN network, the authors extract a central 1/3 area of the rectangle box and use it as the grasp width input.

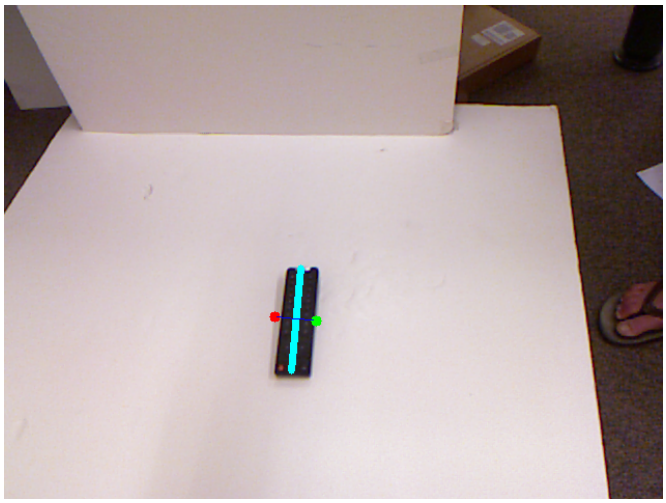

Fig. 2. Re-Label objects using grasp point and grasp width.

During the experiment, we re-labelled the data from the Cornell dataset. To match the input requirements of the network, lines were drawn on the object to represent the grasping area.

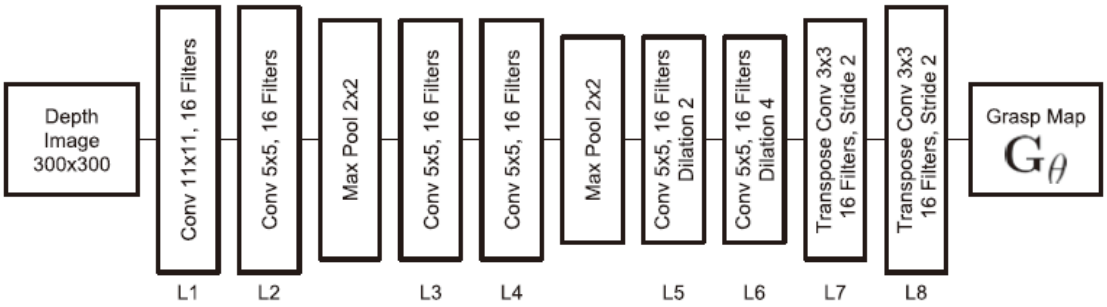

Fig. 3. Network Architecture for GG-CNN2

The width of the line could be adjusted to change the size of the grasping area. The grasping width is always perpendicular to the line representing the grasping area, and its direction can be modified. In addition to drawing the grab line, for round or spherical objects, you can set the grab point and grab radius, that is, the grab Angle is 360° for a particular grab point.

After the object is annotated and the label is obtained, the grasping point and grasping Angle are calculated according to the annotation information, and the grasping width is set and saved as mat type file.

B. Network Architectures and Design

When designing the networks author performed a number of parameter sweeps, varying the number of convolutional filters, filter sizes and dilation parameters in order to find the network design that offers the best performance while still being small enough to allow for real-time inference.They use GG-CNN as a baseline to design an improved network for real-time grasp prediction, which called GG-CNN2. GG-CNN2 is a fully convolutional network based on the semantic segmentation architecture from Yu and Koltun (2016)[12], which uses dilated convolutional layers to provide improved performance in semantic segmentation tasks. The GG-CNN2 uses the same input and outputs as the GG-CNN. To accelerate evaluation, we use the IoU metric for local testing of many different network configurations, and use the SGT simulator for only a smaller subset of well-performing networks. In the experiments, we mainly use GG-CNN2 to train our network shown in figure 3.

V. Experiments

A. Simulation Environment Setup

In pybullet, we constructed a test ground envolves a franka panda robot, a traybox, and an invisible RGBD camera (which is defined using conventions from OpenGL). Befor each grasping task, target objects will be generated somewhere around the world origin.

1) Environment settings: The entire test environment is carried out on a ground palne with the acceleration of gravity set to 9.81; The position where the traybox is placed (0.5, 0, 0); The camera is located at position (0, 0, 0.5) and shoots

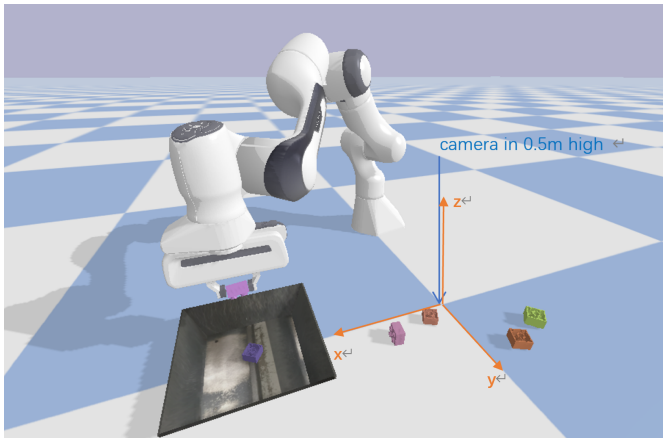

Fig. 4. Simulation Environment Setup in pybullet

```
self.intrinsic_matrix = np.array([[f, 0, self.img_w / 2],
                                  [0, f, self.img_h / 2],
                                  [0, 0, 1]],
                                 dtype=np.float)
self.trans_matrix = get_trans_matrix(euler_angle=[np.pi, 0, 0], pos_offset=self.eye_pos)
```

Fig. 5. Intrinsic and extrinsic matrices definition

in the negative direction of the z-axis; The Panda robotic arm is located at position (0, -0.5, 0); When generating the target objects, the objects fall in random positions and random orientations in the three-dimensional square space of -0.2 to 0.2 on the x-axis, -0.1 to 0.2 on the y-axis, and 0.1 to 0.2 on the z-axis.

2) Camera settings: The camera is placed 0.5m high above the world origin, vertically downward, to acquire images near the origin, where the target objects would be initialized. This pybullet camera is set to captures both RGB and depth images in 480*640 after certain methods to restore from raw information with the help of pybullet api. Thanks to the task is carried out in the simulation environment, since the camera is generated under our wishes, so we can decide everything about the camera, so that the camera calibration work is not necessary during the mapping process from camera image coordinates to world coordinate systems, i.e. from a 2D pixcel point to 3D point in the world coordinates. According to

the camera setting parameters of OpenGL (Pybullet support tinyRenderer and OpenGL to render the simulated world, commonly OpenGL is used for its powerful features. Therefore, pybullet directly uses OpenGL's conventions when defining cameras), we could manually calculate the intrinsic parameter matrix of the camera, and obtain the extrinsic parameter matrix of the camera by the matrix operation of the Euler angle of rpy convention. See Fig.6 for definition of intrinsic and entrinsic matrices.

3) Robot motion settings: All motion control methods of the robotic arm adopt position control, the movement of the robot arm in the gripping task is divided into several phases, each corresponding to a simulation interval in which the end of the robot arm is expected to reach the position and attitude specified at the current stage; The robot's inverse kinematics work is completed by pybullet's built-in inverse kinematics solver. In order to better plan the movement of the robotic arm in this task, the robotic arm is set with seven motion states; First state, the end effector of panda arm reaches the starting position at coordinates (0, 0, 0.2). Second, the end effector move to a place 8 centimeter above the target. Third, the end effector move to the place of the target object waiting for grasping. Fourth and fifth, close the gripper to hold the target object and return to the starting position of the first state. Finally, move the end effector to the position 0.2 m above the traybox and release the target object in the last two state of motion. After this grasping round is completed,

B. Dataset Preperation

In the experiment, we prepare three sets of data to train the ggcnn models respectively.

1) Cornell Dataset: As indicated in the previews section, we use another labelling method to extract the Cornell dataset label information.

2) Lego Dataset: To verify the performance of GGCNN on other datasets, we have decided to generate our own dataset through collecting images in pybullet and and manually labelling. In all, we collected 200 images as well as their labels. Afterward, we evaluated trained GGCNN and GGCNN2 network on this dataset to see how it perform and how it shows similarity between properties illustrated in the original paper. The reasons why we only use images of single lego model would be explained later. In the Pybullet simulation environment, we randomly generate random scalled Lego objects with random poses within a specified area. We recorded and saved RGB and depth images, then carried out the annotation procedures follow the exact same methodology as what we did to Cornell dataset.

To be mentioned with, before creating this Lego dataset, we tried to the dataset with objects provided by pybullet built in package folder named random urdfs. This folder includes 1000 different urdf models, after resizing the model to an appropriate scale to panda robot gripper, we collect 3 images for each model from index 0 to 49, 150 in total, to built our dataset. However after the network training was finished, the final accuracy was as low as about 0.2, which could not converge. After summarizing and reviewing this bad result, we got some experience and reflection as follow.

a) On the one hand, the training set was so small and the objects were so complex in shape that the training process can hardly converge. Each object only covered 3 images and 3 labels, which was not enough to teach a generalized grasping policy to the network. b) On the other hand, the objects from pybullet built in package folder are often of weird shapes, this makes it very difficult to label the grasp information, and it is nearly impossible to ensure the standardization of between different people when labelling, which further makes it difficult for the model to converge.

So we decided to just use Legos of different sizes to make a simple dataset for evaluation. Actually the lego can be seen as three collections, with its side surface up, upward face up and upward face down, which is equivalent to 200 annotations for three sets. At the same time, in order to further ensure the quality of the dataset, we had only one person to carry out all the labeling work, and agree to only label once on the long side when the side surface faces upward, and two vertical labels when upward face up or down, as shown in figure 6.

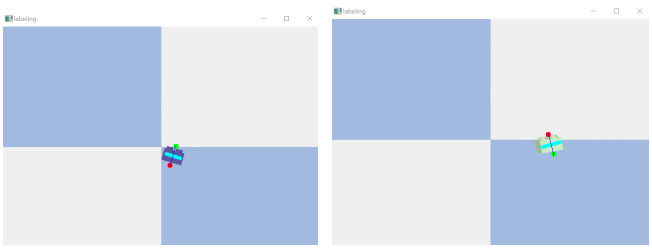

Fig. 6. Examples of labeling our Lego dataset

In our codes, before training, a choice of the network type between GG-CNN and GG-CNN2 is supposed to be made , as well as the training dataset. The GPU used in the experiment is NVIDIA 1660S, and the CPU is AMD 3500X. To maximize training efficiency, the batch size is set to a default value of 40, the learning rate is set to 0.0045, and the weight decay value is set to $1e^{-9}$. The input image size to the network is set to a default of 360x360 pixels. Another option is to include depth images or RGB images as inputs to the network.

C. Model Evaluation

After the trained models are ready, evaluation tasks are designed to test how well the models actually preform in real world (simulation environment). The computer hardware environment we used at this time was a laptop equipped with an AMD R4800u CPU and an AMD Radeon(TM) Graphics integrated graphics card with a memory of memory 256MB.

The evaluation baseline we designed is actually very simple. First, set up multiple rounds of grabbing tasks (L rounds), initialize the scene at the beginning of each round of grabbing tasks, as well ad generate N objects to be grasped near the origin (that is, within the camera's field of view). In a single-round grabbing task, the panda robotic arm performs

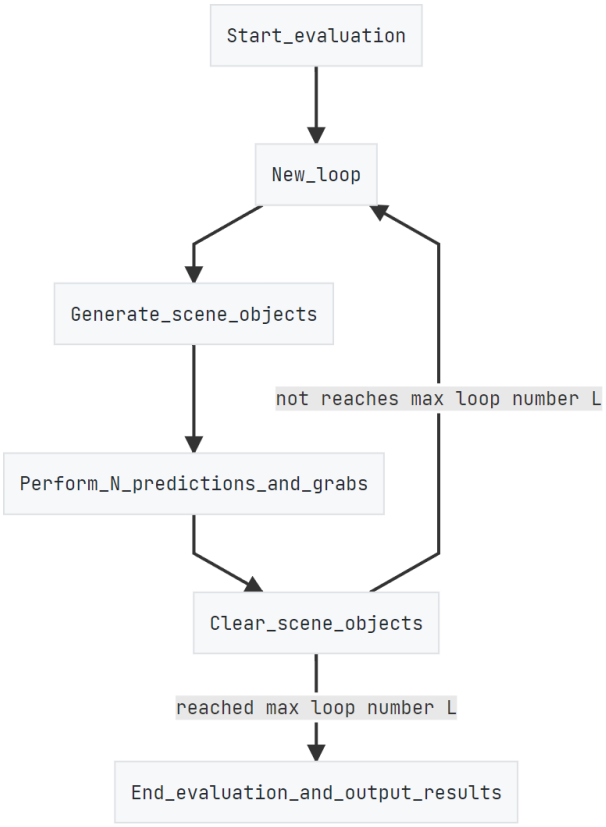

Fig. 7. Evaluation Task Setup

N predictions and grabs respectively. Regardless of whether the N grabs are successful or not, after N grabs, the round is considered to be over, all objects in the current scene would be cleared, then goes into the next round of grabbing, and this cycle is repeated until the completion of the whole L round test. The flow chart is shown in figure 8.

1) Part 1: Evaluation performance of GGCNN and GGCNN2 trained on lego dataset:

We chose to experiment with the network trained on the LEGO dataset first, because on the one hand, this dataset is relatively small and the cost of testing is low, and on the other hand, it can provide experience for the experiment settings of testing for larger networks trained on the Cornell dataset. Since we have manually labeled the lego images we generated in pybullet, we trained a ggcnn and ggcnn2 model to see how they actually perform in a grasping lego task.

In this experimental part, we performed 50 rounds of grabbing in each single experiment, each round in the single experiment generates 3 legos with random positions, random poses and random size (scaled from 1 to 1.5 times) in a given area of the scene, a total of 150 predictions and grabs, and the task success rate and average prediction time are given after each experiment. For GGCNN and GGCNN2, each model performs five such experiments, i.e. each model executes for 750 grasps.

Through the whole test session illustrated in the section

```python
# predict grasp info
start_time = time.time()
pred_info = pred.do_prediction(dep_img, pred_model, model_type)
end_time = time.time()
pred_duration = end_time - start_time
text = "pred duration: {:.3f}s".format(pred_duration)
```

Fig. 8. Method to approximate model prediction duration

above, we obtained the experimental results which are (1) successful grasping rate (number of successful grabs / total number of grabs), (2) average prediction duration (sum of each prediction duration / number of predictions). We believe that the two indicators of successful grasping rate and the average prediction time can reflect the efficiency of the model. The successful capture rate comprehensively reflects the quality of the labeled dataset and the effect of model training on the performance of the actual robot when performing tasks in real world (here is the pybullet simulation environment). The average prediction duration reflects the time economy of the model in practical application, and determines how well it can perform in real-time prediction capability the model can achieve.

When evaluating the average prediction duration, the method we used was to obtain start time and end time with help of python built-in method time.time(), then express the duration with start-end time difference. This approach may not be very rigorous, but it provides data that allows us to compare the performance of different models. The Python code description of this method is shown in figure 9.

At the stage of image generation for the Lego dataset, legos were specified to be generated randomly in the region of x range [-0.20, 0.20], y [-0.10, 0.20], z [0.05, 0.15], but after some simple tests we found a feature of GGCNN that it is very sensitive to the labelled information, which makes the GGCNN has strong prediction biases caused by two dimensions. (1) Because of the perspective relationship in the photos obtained from the camera, when we label a picture with a grasping information perpendicular to the ground direction with a perspective relationship, there is a bias in the labelled information due to the perspective when labelling objects that are off the center of the image. And as a result, the model has learned this bias and performs poorly in predicting the grasping information of objects that are off the center of the image. On the contrary, the model has good prediction results for objects closer to the image center. (2) When the objects in the capture image appear in places where the objects in the test set did not appear during training, the trained model would not perceive the existence of any objects in the image due to the trained CNN weights. This feature brings restriction to the position of the object to be grasped, and the position of the object to be grasped in the captured image should fall in the places where the trained model has seen in the training set.

When doing experiments on GGCNN and GGCNN2 trained

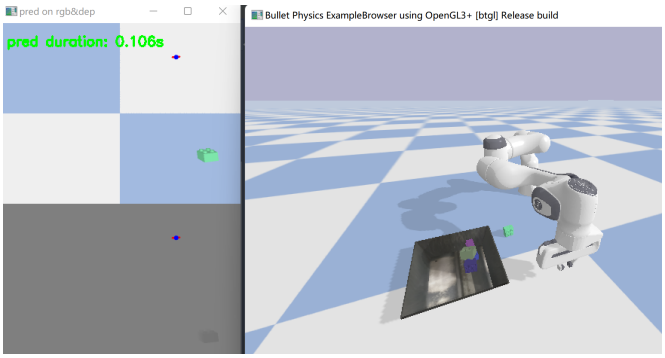

Fig. 9. Example of prediction bias on object off the center of the captured image

```
prediction_info[3] += 10  # Increase 10 pixels to the predicted grasp width
# reach the object position and wait for closing gripper
pos[2] = pos[2] / 2  # Here a cusomized operation to determine z coordinate of grasp point
```

Fig. 10. Customized operations we did

on our Lego dataset, we specified the test lego to be generated in a smaller x-y region (x [-0.15, 0.15] y [-0.1, 0.1]). For the grasp width, in order to give a certain degree of redundancy, we increased the predicted grasp width in pixel by 10 pixels before executing grasp task. In addition, when converting the predicted 2d information to 3d space, the depth of the predicted grasp point was based on the depth information extracted from the captured depth map, which gave us some extents of customized adjustment. In this task, we set the transformed 3d grasp point's z-axis position as half of the corresponding predicted point's z-axis coordinate obtained directly from captured depth image. The operations in code are shown in figure 10.

The result of experiment Part I is shown in Table I.

TABLE I
EXPERIMENT RESULTS

MODEL	SUCCESSFUL RATE	TIME
GGCNN2-Lego	92.6667%	122.5904 ms
	94.6667%	111.8193 ms
	96.6667%	108.6800 ms
	92.0000%	131.9739 ms
	92.3333%	109.1270 ms
Average	93.7000%	116.8380 ms
GGCNN-Lego	88.6667%	94.9162 ms
	83.3333%	108.0038 ms
	82.6667%	108.8400 ms
	80.6667%	95.3636 ms
	84.3333%	98.4625 ms
Average	83.9000%	101.1170 ms

According to the experimental results, GGCNN2 shows a better experimental success rate than GGCNN, with a success rate excess of about 10%. At the same time, GGCNN2 is not as fast as GGCNN, and the average prediction duration is about 10 to 20 milliseconds longer than GGCNN. Due to the different hardware environment and test environment of the experimental computer, the above experimental results did not

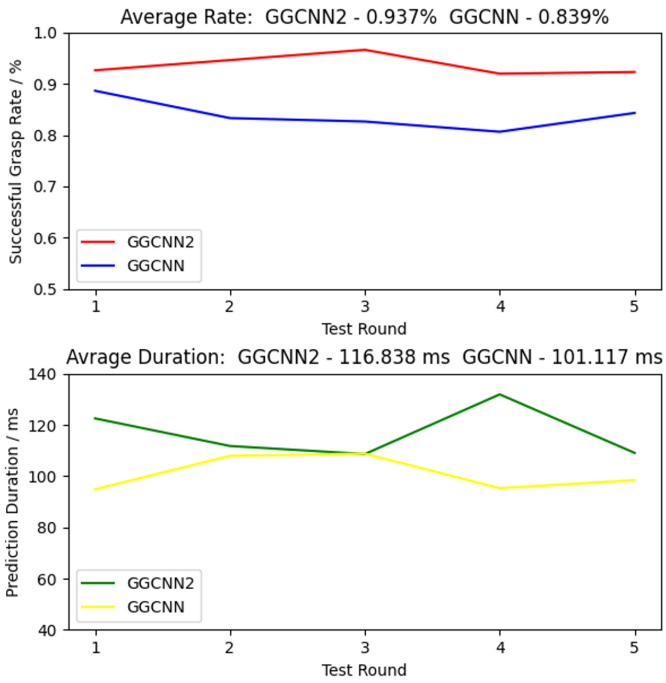

Fig. 11. Comparison of line charts of Part I experimental results

achieve the real-time prediction grasping of 50Hz described in the original article, but it solidly confirms the description of GGCNN and GGCNN2 in the original article, that is, GGCNN2 has better prediction performance than GGCNN, as well as GGCNN is lighter and has a faster prediction speed.

2) Part 2: Evaluation performance of GGCNN and GGCNN2 trained on Cornell dataset:

In order to reasonably validate the real performance and explore the model generalization of our trained GGCNN models on the Cornell dataset, we need to more reasonably set up the experiments and carefully select representative experimental objects to be tested. In the experiments of Part 2, we chose the evaluation set of EGAD dataset as the object to be tested for grasping experiments. EGAD is an evolved grasping analysis dataset for diversity and reproducibility in robotic manipulation. This work was accepted by IEEE RA-L, April 2020.[8]

The EGAD evaluation set is labelled in a 7x7, one dimension represents the shape complexity and illustrated by integers from 0 to 6, and the other dimension represents the difficulty of grasping, illustrated by latters from A to G. As shown in figure blown, the A0.obj is the simplest, easiest to grasp object, through to G6.obj, the most complex, difficult to grasp object.

Firstly, when preparing the evaluation objects, since the shape of the test set objects may not match the width of the Panda robot gripper, we first scaled the obj formatted objects according to the maximum width of the gripper, i.e. 8cm. The collision model needed in the simulation was then generated in batches. In the actual generation of the objects in pybullet, the processed models was scaled by a factor of 0.6 to ensure that the objects were within a reasonable size range relative

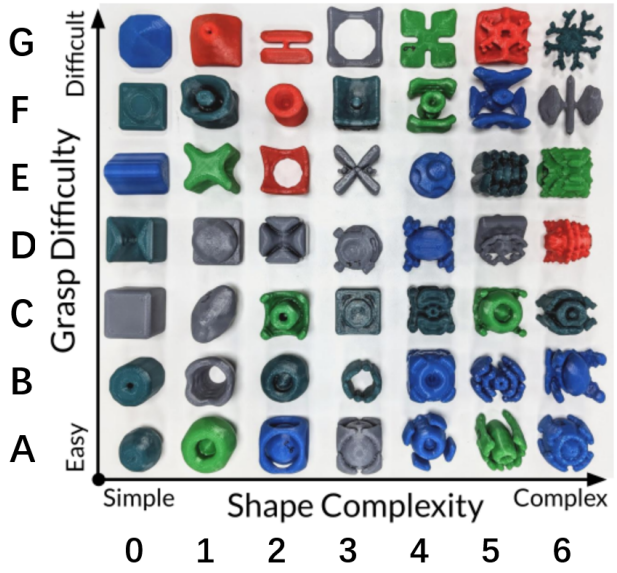

Fig. 12. EGAD evaluation set visualization

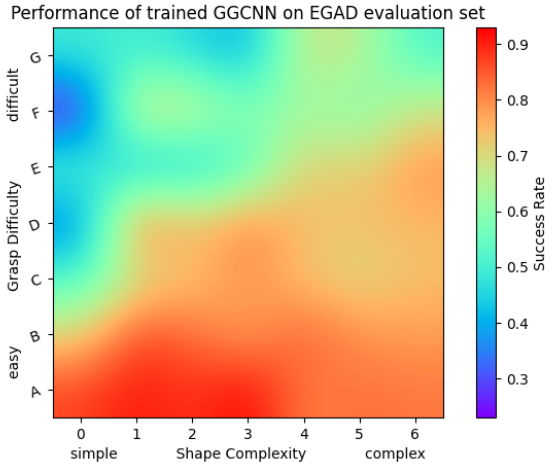

Fig. 13. Visualization for results of trained GGCNN

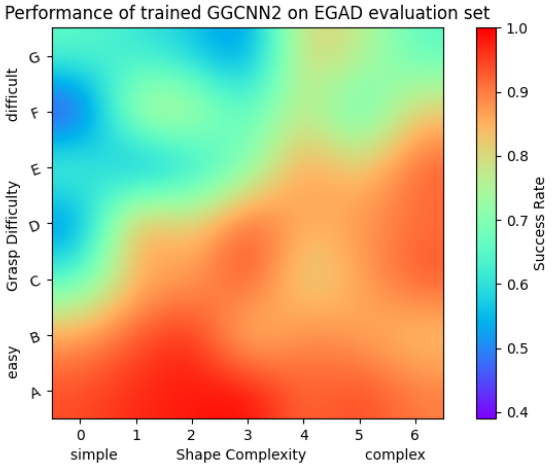

Fig. 14. Visualization for results of trained GGCNN2

to the gripper.

In this experiment, the trained GGCNN and GGCNN2 models executed 100 predictions and graps for each object in the evaluation set, i.e. each model performed 4900 grasps, and the grasping success rate corresponding to each object was obtained separately after the test was completed, after which the results were plotted for comparison. Similar to Part 1, we did the same operations, increasing the predicted grasp pixel width by 10 pixels and approximating the position on z-axis of the transformed 3d grasp point as one-half of the position on z-axis coordinate corresponding to the predicted point location.

When visualizing the results, since each model only gots 49 data, the results are discrete in the plots, and in order to better visualize how the model performs on the EGAD test set, we used bicubic interpolation to give the plots a better fluency.

According to the plotted results, GGCNN and GGCNN2 have very similar performance, and GGCNN2 shows slightly higher grasping success rate than GGCNN overall. Both of them showed an remarkable ability to predict the grasp information to object of low shape complexity and low grasp difficulty.

Intuitively, the lower left corner of the plot should have a higher success rate, and the experimental results matched well. Similarly, the upper right corner were expected to have the lowest success rate, but the experimental results were on the contrary. As grasp difficulty increases, the grasping success rate has a certain degree of improvement with the increase of shape complexity instead.

After some analysis, the GGCNN model only predicts on the captured 2d images and is not involved at the conversion of 2d grasp information to the actual 3d grasp information, i.e., the GGCNN is not involved in the prediction of depth, which leads to the unreliability of the GGCNN model in depth prediction in the actual grasping task. For objects with high grasping difficulty and low shape complexity, the grasping depth obtained by the GGCNN and the preset depth conversion method is often unreliable.

Take an instance, for object from D0 to D6, D0 has medium grasp difficulty and lowest shape complexity, but is hard to grasp for both GGCNN and GGCNN2. Object D0 is shown in below, what can be seen for D0 is that there is a hollow area in the middle of the object, which leads to the prediction that the grasping point is likely to fall in this hollow area. After the prediction of GGCNN, with our preset simple depth conversion method, the z-axis coordinate of the converted 3d grab point is undoubtedly 0, which is on the ground plane and obviously unreasonable. With the increase of shape complexity, the hollow area decreases and finally would vanish, and the successful grasp rate increases consequently.

As the shape complexity increases, the hollow regions are decreasing, which reduces the error rate of depth prediction. Therefore, grasping objects with hollow regions is a disaster

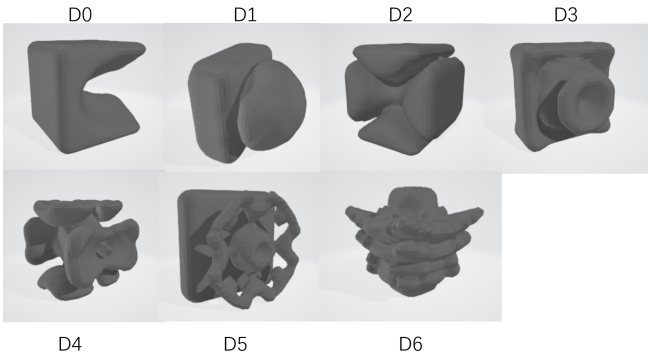

Fig. 15. Visualizations from object D0 to D6

for the GGCNN model. Intuitively, the most straightforward way to compensate for this drawback is to design a more general depth conversion method. However, this task is another hardcore research topic.

VI. Conclusion

According to our experimental results, GGCNN and GGCNN2 can basically perform fast and effective prediction of planar grasping poses from a depth image of a pixelwise basis. GGCNN2 has relatively better prediction performance, while GGCNN is lighter and faster. In our experiment setting, our trained GGCNN and GGCNN2 require around 100ms to execute a single prediction, and both have good task successful rate, where GGCNN achieved an average of 83.9% and 93.7% for GGCNN2. Thus, when taking into consideration the significant enhancement in accuracy, the supplementary computational time and resources consumed by GGCNN2 can be regarded as insignificant.

However, there are still flaws in GGCNN.

1) First: GGCNN has extremely high requirements for the labeling quality of the training dataset. This point is concluded from our process of preparing the LEGO dataset. In order to ensure the quality of dataset annotation, the biases should be avoided as much as possible. When labeling the image, bias introduces by perspective relationship in the picture (the object should be in the center of the picture as much as possible) and the bias caused by the labelling people. In our work, a dataset of 200 high-quality labeled image was enough to teach the model to learn a good grasp prediction ability, which also shows that GGCNN has a good convergence ability under the condition of a small dataset.

2) Second: Through our experiment, we have discovered that since the GGCNN family predicts 2d pixel grasp information, the 3d grasp depth obtained according to the GGCNN model is often unreliable when predicting the grasp poses of objects with hollow structures. We hypothesize that this is due to the inherently limited information that can be obtained from a single perspective of an object. As a result, for 3D objects with unconventional convex shape-alike structures (such as a torus), the predicted grasp quality may not be optimal.

To address this issue, the original authors proposed a solution called multi-viewpoint grasping. However, we believe that this approach is too expensive and there may be other solutions. Here, we propose two constructive suggestions:

(1) Integration of a target identification and localization module like YOLO to avoid prediction bias resulting from disparities in perspective and achieve real-time grasp prediction after YOLO localization. This approach has been supported by a 2023 study[5].

(2) Integration of a more widely applicable and generalized depth transformation module to overcome GGCNN's catastrophic predictions for objects with hollow structures, thereby enhancing its generalizability.

Overall, GGCNN is a sufficiently fast and accurate approach to selecting grasp points for previously unseen items. Enabling robots to function as an active component in a closed-loop grasping pipeline which plays a vital role in the development of resilient and dependable robotic systems.

Acknowledgments

We would like to express sincere appreciation to Professor Song for his invaluable assistance and support throughout our course project and writing process. We learned a lot about methodologies of academic research and writing from him. Without the assistance of Professor Song, this work wouldn't be possible. Additionally, we would like to thank Wang Dexin, a graduate student in Shandong University, who had shared valuable knowledge about GGCNN on the Internet.

References

[1] Jeannette Bohg et al. "Data-Driven Grasp Synthesis - A Survey". In: *IEEE Transactions on Robotics* 30.2 (2014), pp. 269–288. DOI: 10.1109/TRO.2013.2294866.

[2] Minghui Jiang, Samuel Moseson, and Ashutosh Saxena. "Efficient grasping from RGBD images: Learning using a new rectangle representation". In: *The International Journal of Robotics Research* 30.2 (2011), pp. 199–212.

[3] Edward Johns, Stefan Leutenegger, and Andrew J Davison. "Deep learning a grasp function for grasping under gripper pose uncertainty". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4461–4468.

[4] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 705–724.

[5] Xu Li et al. "A YOLO-GGCNN based grasping framework for mobile robots in unknown environments". In: *Expert Systems with Applications* 225 (2023), p. 119993. DOI: 10.1016/j.eswa.2023.119993.

[6] Jeffrey Mahler and Ken Goldberg. "Learning deep policies for robot bin picking by simulating robust grasping sequences". In: *Proceedings of the Conference on Robot Learning (CoRL)*. 2017, pp. 515–524.

[7] Jeffrey Mahler et al. "Dex-Net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2017.

[8] Philip Morrison, Peter Corke, and Jürgen Leitner. "EGAD! An Evolved Grasping Analysis Dataset for Diversity and Reproducibility in Robotic Manipulation". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4368–4375. DOI: 10.1109/LRA.2020.2992195.

[9] Philip Morrison, Peter Corke, and Jürgen Leitner. "Learning robust, real-time, reactive robotic grasping". In: *The International Journal of Robotics Research* 39.2-3 (2020), pp. 183–201. DOI: 10.1177/0278364919859066.

[10] Tow Morrison et al. "Cartman: The Low-Cost Cartesian Manipulator that Won the Amazon Robotics Challenge". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7757–7764. DOI: 10.1109/ICRA.2018.8463191.

[11] Philipp Rubert et al. "On the relevance of grasp metrics for predicting grasp success". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 265–272. DOI: 10.1109/IROS.2017.8202167.

[12] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *International Conference on Learning Representations (ICLR)*. 2016.