applied sciences

Article

Task-Specific Grasp Planning for Robotic Assembly by Fine-Tuning GQCNNs on Automatically Generated Synthetic Data

Artúr István Károly * and Péter Galambos

Antal Bejczy Center for Intelligent Robotics, Óbuda University, Bécsi út 96/B, 1034 Budapest, Hungary
* Correspondence: artur.karoly@irob.uni-obuda.hu

Abstract: In modern robot applications, there is often a need to manipulate previously unknown objects in an unstructured environment. The field of grasp-planning deals with the task of finding grasps for a given object that can be successfully executed with a robot. The predicted grasps can be evaluated according to certain criteria, such as analytical metrics, similarity to human-provided grasps, or the success rate of physical trials. The quality of a grasp also depends on the task which will be carried out after the grasping is completed. Current task-specific grasp planning approaches mostly use probabilistic methods, which utilize categorical task encoding. We argue that categorical task encoding may not be suitable for complex assembly tasks. This paper proposes a transfer-learning-based approach for task-specific grasp planning for robotic assembly. The proposed method is based on an automated pipeline that quickly and automatically generates a small-scale task-specific synthetic grasp dataset using Graspit! and Blender. This dataset is utilized to fine-tune pre-trained grasp quality convolutional neural networks (GQCNNs). The aim is to train GQCNNs that can predict grasps which do not result in a collision when placing the objects. Consequently, this paper focuses on the geometric feasibility of the predicted grasps and does not consider the dynamic effects. The fine-tuned GQCNNs are evaluated using the Moveit! Task Constructor motion planning framework, which enables the automated inspection of whether the motion planning for a task is feasible given a predicted grasp and, if not, which part of the task is responsible for the failure. Our results suggest that fine-tuning GQCNN models can result in superior grasp-planning performance (0.9 success rate compared to 0.65) in the context of an assembly task. Our method can be used to rapidly attain new task-specific grasp policies for flexible robotic assembly applications.

Keywords: grasp planning; robotic grasping; robot manipulation; deep learning; GQCNN; synthetic data

check for updates

Citation: Károly, A.I.; Galambos, P. Task-Specific Grasp Planning for Robotic Assembly by Fine-Tuning GQCNNs on Automatically Generated Synthetic Data. *Appl. Sci.* **2023**, *13*, 525. https://doi.org/10.3390/app13010525

Academic Editor: Tae-Seong Kim

Received: 20 December 2022
Revised: 26 December 2022
Accepted: 27 December 2022
Published: 30 December 2022

1. Introduction

Robots are excellent at performing tasks that would be too tedious and repetitive for human workers. As a result, they are widely used in industrial packaging and assembly processes, where their ability to reliably and rapidly repeat the same tasks for long durations can be well-utilized. However, modern trends in the industry tend to shift toward flexibility, bringing forth new requirements for robot applications and giving rise to novel challenges in robotic manipulation. One such challenge is grasp planning, which considers finding appropriate grasps for a given object according to a quality measure, such as analytically computed wrench-space metrics [1], empirical evaluation on physical trials [2] or similarity to human-provided grasps [3].

Grasp planning is needed because the pose and/or the object's geometry is often not known in advance. Thus, a "rigid" robot program that can only repeat the exact same movements over and over is not sufficient for performing robotic manipulation in such tasks. By utilizing grasp planning in the robot manipulation pipeline, the application can quickly and easily be adjusted for novel objects or objects with unknown poses.

Determining the quality of grasps is a complex challenge since it greatly depends on the object geometry, the gripper geometry, dynamic properties, such as friction forces, and the task that will be performed after the object is grasped [4]. Analytical approaches can offer reliable solutions, based on simulated contacts and dynamics. Still, they often rely on known object geometry, material, inertia matrix, etc., and their computation is often time-consuming [1]. Object detection and pose estimation also have to be performed on top of the analytical grasp planning for objects with unknown poses. Additionally, the relevant information about the task which will be performed after the grasping is very hard to formulate, and thus, it is nearly impossible to utilize it in grasp planning with analytical approaches. On the other hand, data-driven approaches for grasp planning can incorporate object detection, pose estimation and grasp planning into a single model [5]. With the use of large datasets, they are able to learn general grasping policies, which can be applied to novel objects as well [2,5]. Grasp prediction with such models is also significantly faster when compared to analytical methods [5]. However, a large dataset is required to train such models. Collecting a large-scale grasping dataset using physical robots can take up to months and incur great costs [2]. As a result, offline-generated simulation-based synthetic data are usually preferred, where the analytical grasp quality approaches can be utilized for ground-truth generation [5].

Miller and Allen created a simulator called Graspit! for evaluating robotic grasps [1]. In Graspit! , arbitrary gripper and object geometries can be imported. Using a sampling-based method, a large number of grasps can be evaluated for a given gripper-object combination to find the best-quality grasps. It can also be used to simulate and compute the quality of grasps in a dynamic environment, but this is out of the scope of this paper. Goldfeder et al. created the Columbia Grasp Database and a corresponding data-driven approach for grasp planning with the help of the Graspit! simulator [6,7]. Their approach is based on matching partial object geometry information against a large dataset of 3D models and selecting appropriate grasps from a set of pre-computed grasps for the matched objects. A similar approach of using pre-computed grasps, called Dex-Net 1.0, was introduced by Mahler et al. who leveraged the benefits of cloud computing to reduce the application run-time significantly [8]. For measuring the similarity between objects, they used multi-view convolutional neural networks. Later, based on their results in Dex-Net 1.0, Mahler et al. compiled a huge synthetic dataset (called Dex-Net 2.0) of 6.7 million point clouds with grasps and associated analytic grasp metrics, where grasps are represented by a planar position, an angle and the depth of the gripper relative to the RGB-D sensor [5]. They also proposed a convolutional neural network architecture called grasp quality convolutional neural network (GQCNN) that could predict the probability of the success of grasps directly from depth image data. They showed that a GQCNN trained on their Dex-Net 2.0 synthetic dataset outperformed other state-of-the-art approaches using point cloud registration while being 3× faster. They also demonstrated great precision (99%) on a set of novel objects.

One drawback of the previously mentioned methods is that they only consider grasping in isolation. However, the quality of a grasp also greatly depends on the task which will be performed after the grasping is done. Costanzo et al. demonstrated that fixed grasps can make a robot manipulation task infeasible [9]. They suggest, that in-hand manipulation maneuvers, such as object pivoting based on tactile feedback (described in [10] in detail) might be necessary for a successful task solution, and as such, grasps should be selected, taking this into consideration. On the other hand, task-specific grasp planning approaches aim to overcome this challenge by incorporating a task encoding into the grasp prediction pipeline [11–13]. The task encoding in these approaches is usually categorical, and describes a generalized use case such as poke, pour water, etc. While these solutions work well for some general scenarios, categorical task encoding might not be sufficient in specific pick-and-place tasks for robotic assembly, where object geometries and assembly order limit the number of appropriate grasps. When approached with a categorical task encoding, such solutions would have to represent each unique assembly as its own category. Since the number of categories and their semantic meaning are core components of such networks, the model would have to be retrained from scratch each time a new assembly task is available. We suggest that a transfer-learning-based approach is much better suited for such scenarios.

Transfer learning allows for reusing large deep-learning models which were pre-trained on large datasets. During transfer learning, only the top layers of the network are modified, and the lower layers, which extract more general features, are left untouched [14]. Since only a fraction of the network's parameters needs to be adjusted, a relatively small dataset can be used without overfitting. This process is called fine-tuning the network. This paper proposes an automated synthetic dataset generation pipeline using Grasipt! and Blender for fine-tuning GQCNN models for task-specific grasp planning in robotic assembly. We focus on predicting geometrically well-placed grasps and do not examine dynamic grasp qualities. The results are demonstrated and discussed through a simple, yet representative simulated task.

2. Methodologies

Similarly to the Columbia Grasp Database by Goldfeder et al. our synthetic dataset generation pipeline also utilizes Graspit! for determining grasps qualities. Figure 1 shows the proposed synthetic dataset generation pipeline. As it can be seen, Graspit! is used to automatically acquire grasps for the object in two scenarios. The first one is the pick scenario, from where Graspit! provides multiple possible grasps for picking the object. The second scenario is the placing, where Graspit! provides grasps which are suitable for placing the object. For our experiments, we use a Franka Emika Panda robot, its signature parallel jaw gripper, and an object made up of simple box primitives. Our experiment mimics an insertion-type assembly subtask, where the object should be grasped at the thicker part. By chaining such pick-and-place setups according to the assembly order one after another, a more complex assembly could also be similarly composed. The required inputs in this setup are the 3D models of objects, the gripper model, and knowledge of the assembly process (location of parts relative to each other and assembly order).

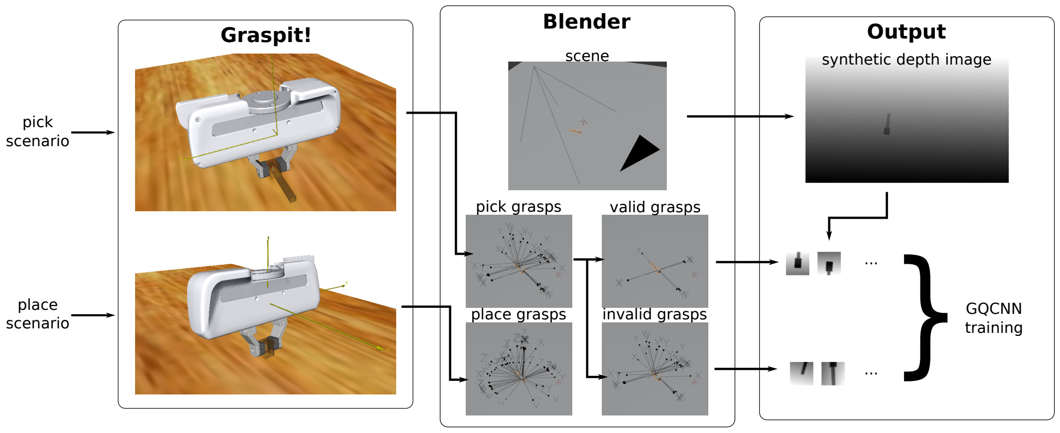

Figure 1. Synthetic data generation and GQCNN training pipeline.

The here-discussed pipeline incorporates Graspit! with the robot operating system (ROS), an open-source robotics framework that provides useful tools for creating and managing robotic applications [15]. With the help of the Graspit! ROS interface (Available online at https://github.com/graspit-simulator/graspit_interface, accessed on 12 December 2022) in conjunction with Graspit! Commander (Available online at https://github.com/graspit-simulator/graspit_commander, accessed on 12 December 2022), one can automate the Graspit! grasp planning process using Python programming language and ROS. We store the outputs of the Graspit! grasp planning in multiple JSON files, where the name of the JSON file reflects whether the contained grasps are for the pick or the place scenario. The files contain the object pose, the list of search energies from Graspit! (these can be used to make a hierarchy of predicted grasps), and a list of predicted grasp poses. Both the object pose and the grasp poses are relative to the world frame, and they are made up of a 3D vector for object/gripper location and a quaternion (in $[x, y, z, w]$ format) for object/gripper orientation.

For generating the synthetic depth images, we use Blender, an open-source 3D computer graphics software [16]. Blender provides a Python API that offers programmatic control over the rendering and the scene, which our solution takes advantage of to automate all the necessary processes inside Blender. Through this API, we modify the intrinsic and extrinsic parameters of the camera in Blender so they match the "real-world" calibrated camera. In our experiments, we use the camera parameters for the PrimeSense camera provided with the GQCNN implementation (Available online at https://github.com/BerkeleyAutomation/gqcnn, accessed on 12 December 2022). After the camera setup, the grasp poses which were saved from Graspit! are loaded, and all the necessary transformations are performed on them to represent all the pick and place grasps relative to the object frame in Blender, regardless of its placement in Graspit! and Blender. After this, based on geometric proximity, the pick grasps are sorted into two groups. The group of pick grasps with a corresponding place grasp is classified as valid, while the pick grasps with no corresponding place grasps are classified as invalid. Figure 2 displays the pick grasps, the place grasps, and the separation of the pick grasps into valid and invalid grasps. The grasps are represented as coordinate frames in Blender. In our experiments, two grasps are considered "corresponding" if the distance between their position is smaller than 5 mm and their orientation forms a smaller angle than 10°. Based on this correspondence, the grasps are also pre-filtered to remove pick grasps that are too similar. Once all the pick grasps are classified as either valid or invalid, a JSON object is generated and saved, which contains a list of 2D grasps, represented by a 2D point (in pixel units), an angle, and a label (1 for valid grasps and 0 for invalid grasps). After this, an RGB-D image is rendered. The RGB image is saved as a PNG, while the depth data are saved as a NumPy array (npy file). We also generate segmentation masks for each rendered frame using the Blender Annotation Tool (BAT) (Available online at https://github.com/ABC-iRobotics/blender_annotation_tool, accessed on 12 December 2022), although currently these segmentation masks are not utilized [17]. The Blender scene is prepared so that the object's location along the X and Y axes and its orientation along the Z axis are set to a random value for each frame. As a result, repeating the above process for multiple frames leads to a collection of rendered synthetic depth images and corresponding 2D grasps for multiple object poses.

Using the output from Blender, a Python script creates a synthetic grasp dataset in Dex-Net 2.0 format. The Python script loads the depth images and the corresponding 2D grasps and crops, rotates and resizes the depth image to create 32×32 sized depth images for each grasp, where the grasp point is in the middle of the image, and the opening direction of the gripper is horizontal. Since we only use grasps which are appropriate for picking (but not necessarily for placing), additional 12 invalid grasps are added to each grasp (both valid and invalid), which are located at the same spot where the original grasp is, but they are rotated in either the positive or negative direction along the Z axis in increments of 15° (up to $+90°$ and $-90°$). During our experiments, adding these additional invalid grasps improved the accuracy of the orientation of the predicted grasps significantly. For cropping, we found that a 128×128 square proved to be better than 64×64, although the best value most likely depends on the specific setup (camera-object distance). The 32×32 depth images are bundled in a single NumPy array and saved as an npz file. Similarly, the labels and the hand poses (containing the depth of the grasp points) are saved as npz archives.

Finally, the GQCNN, which was pre-trained on the Dex-Net 2.0 dataset, is fine-tuned on our automatically generated synthetic grasp planning dataset, using the scripts provided for fine-tuning with the GQCNN implementation (Available online at https://github.com/BerkeleyAutomation/gqcnn, accessed on 12 December 2022). During fine-tuning, a train/validation split of 0.9/0.1 was used and trained for 60 epochs. We used a base learning rate of 0.01 with sparse loss, momentum-based optimizer, and 0.999 as the decay rate. All the specific configuration settings can be found in our supplementary materials.

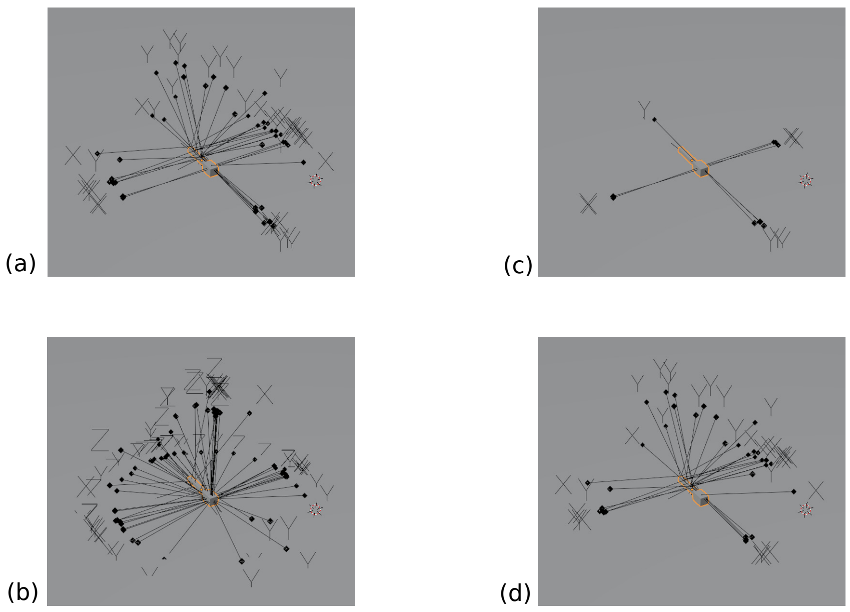

Figure 2. Proximity-based classification of pick grasps visualized inside Blender: (**a**) pick grasps, (**b**) place grasps, (**c**) valid grasps out of all the pick grasps, (**d**) invalid grasps out of all the pick grasps.

After training, we evaluated the GQCNN models using synthetic depth images from Blender and the Moveit! task constructor (MTC) framework [18]. MTC provides a flexible way for defining a complex task, such as our pick-and-place experiment, using a modular approach where elementary subtasks (called stages) can be used as building blocks to compose the whole task. MTC can perform the robot motion planning for the whole task, while also considering collisions in the scene. We define a grasp generator stage that queries a single best grasp from a GQCNN, given a depth image, and using the ROS-based GQCNN grasp planning service from the GQCNN implementation. With this new generator stage, we re-purpose the MTC pick-and-place demo scene to conduct our experiments. The demo scene uses the Franka Emika Panda robot. We add the table, the object, and the camera in the MTC scene in the same relative arrangements as in Blender. Using the best predicted grasp from the GQCNN model, we perform the robot motion planning using MTC and evaluate the GQCNN models according to the motion planning results. The collision checking is carried out throughout the whole task for the whole robot arm. During the evaluation, a fixed place pose is defined for the object. MTC can identify which stage of the task execution resulted in failure. Grasps resulting in a collision between the robot hand or robot arm and the table at the time of placing the object (during the solution of the inverse kinematics computation stage) are considered failed grasps, while grasps that do not result in such collisions are considered successful. It is important to note that this evaluation only considers geometric criteria in the form of collisions, and it has no concern about the quality of the grasps regarding force closure or other dynamic properties. We hypothesize that since the pre-trained GQCNNs already consider dynamics and the positive grasps in our dataset are generated from Graspit! which also provides robust grasps, the predicted grasps with our fine-tuned GQCNN models will also be robust. This statement, however, needs to be validated by real-world trials in the future.

In summary, the main components of the synthetic data generation pipeline, GQCNN training, and evaluation process, with their purpose, and requirements, are as follows:

- **Graspit!**: Used for automated robust grasp generation, given two scenarios, the picking, and the placing scenario. The 3D models of the robot gripper, the object, and the table are needed for both scenarios. In the picking scenario, the object must be in a natural, stable lying position on the table. In the placing scenario, the object must be in the pose it should have after the assembly is complete, relative to all the other relevant (previous, according to the assembly order) assembly parts. The generated grasp poses for both scenarios are expressed as the 3D poses of the robot gripper's frame relative to the Graspit! world frame.

- **Blender**: Used for rendering synthetic RGB-D frames and ground-truth grasp information for training the GQCNN models and for evaluation. The Blender scene must contain a camera and 3D models for the object and the table. For each newly rendered frame, the object must be in a randomized natural lying pose on the table within the field of view of the camera (position along the X and Y axes and rotation around the Z axis of the Blender world frame are randomized). The Blender camera's intrinsic parameters must match the intrinsic parameters of the camera used in the real-world setup. These parameters can be determined by camera calibration. The effects of different extrinsic camera parameters during training and inference are discussed in Section 3.3. The pick and the place grasps from Graspit! are all transformed into a common coordinate frame inside Blender (the object's frame), and the pick grasps are classified as either valid or invalid, based on their proximity to place grasps. The valid and invalid grasp points are projected onto the image plane using Blender's camera projection. Together with their orientation, they are used to create ground-truth grasp information for training the GQCNN models.
- **Moveit! Task Constructor**: Used for automating the evaluation of the GQCNN models after they have been trained. It uses the robot model, the 3D model of the object and the table, and a dummy object (cylinder) for the camera. The camera, the object, and the table can be placed at any location in the robot's workspace, but their poses relative to each other must be the same as in Blender at the time the depth image was rendered that the model is currently evaluated on. MTC performs motion planning for the whole task with collision checking, using the grasp predicted by the GQCNN and a fixed place pose. The evaluation is considered successful if the motion planning for the task can be completed successfully and unsuccessful if the motion planning results in failure due to a collision during the solution of the inverse kinematics solution for the place pose.

Figure 3 shows the flowchart representation of the automated synthetic grasp dataset generation.

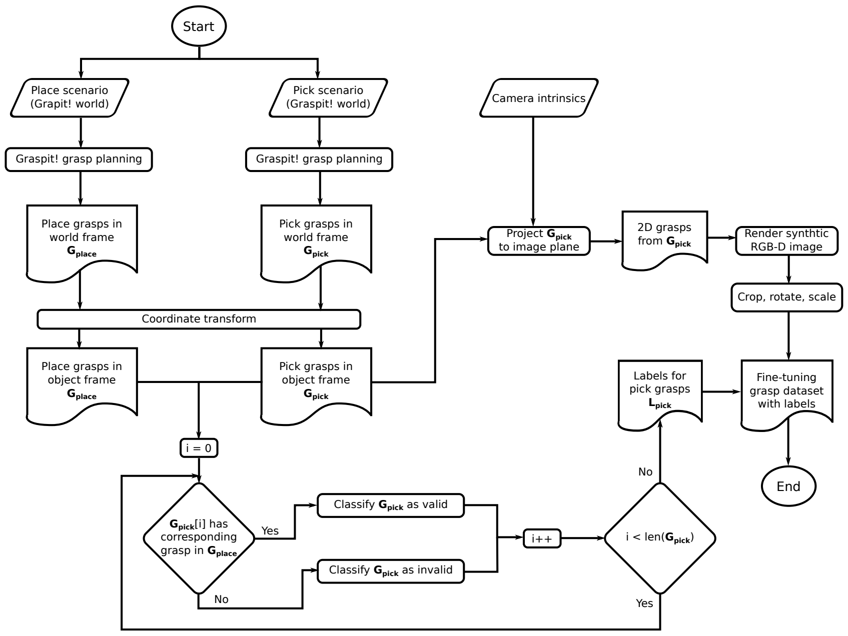
Start
Place scenario (Grapit! world)
Pick scenario (Graspit! world)
Camera intrinsics
Graspit! grasp planning
Graspit! grasp planning
Place grasps in world frame $\mathbf{G}_{\mathbf{place}}$
Pick grasps in world frame $\mathbf{G}_{\mathbf{pick}}$
Project $\mathbf{G}_{\mathbf{pick}}$ to image plane
2D grasps from $\mathbf{G}_{\mathbf{pick}}$
Render synthtic RGB-D image
Coordinate transform
Crop, rotate, scale
Place grasps in object frame $\mathbf{G}_{\mathbf{place}}$
Pick grasps in object frame $\mathbf{G}_{\mathbf{pick}}$
Labels for pick grasps $\mathbf{L}_{\mathbf{pick}}$
Fine-tuning grasp dataset with labels
i = 0
No
End
$\mathbf{G}_{\mathbf{pick}}$[i] has corresponding grasp in $\mathbf{G}_{\mathbf{place}}$
Yes
Classify $\mathbf{G}_{\mathbf{pick}}$ as valid
i++
i < len($\mathbf{G}_{\mathbf{pick}}$)
Classify $\mathbf{G}_{\mathbf{pick}}$ as invalid
No
Yes

Figure 3. Flowchart of the automated synthetic grasp dataset generation procedure.

3. Results

3.1. Experimental Setup

In our experiments, a simple peg-in-hole-like insertion task is considered, where we evaluate the quality of predicted grasps from different GQCNN models according to the success rate of collision-free motion plans. The experimental setup can be seen in Figure 4 (as visualized by RViz). The peg object in the experiments is made up of two box primitives, one of which is sized 10 cm in the X direction and 1 cm in both Y and Z directions, with the origin being the geometric center of the box, the other one is sized 3 cm in X and 2 cm in Y and Z directions and is located 3.5 cm in the X direction. In the setup, the object is lying on its side (the Z axis of the object frame pointing upwards) on the surface of a table. The table is simply represented as a flat surface.

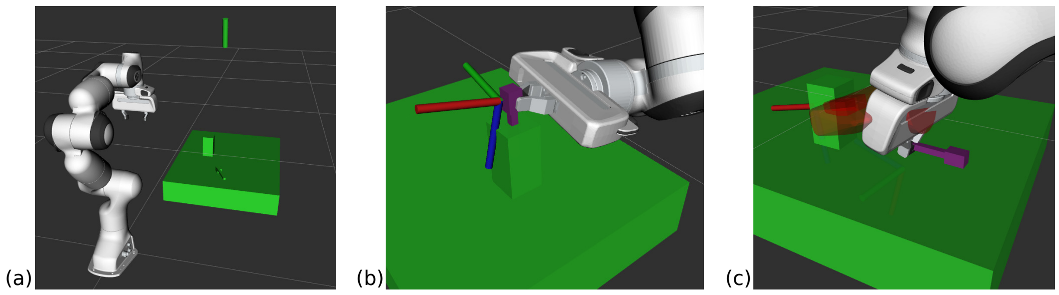

Figure 4. Experimental setup used for evaluation: (**a**) scene setup with the robot, object camera, and table, (**b**) planned grasps which do not result in collisions are evaluated as successful, (**c**) grasps that result in collisions between the robot and the table are evaluated as unsuccessful.

Since the 3D grasp pose predicted by the GQCNNs is oriented according to the line between the depth camera and the grasp point, we included an additional platform for placing in MTC to avoid unnecessary collisions between the robot arm and the table. Inside Graspit! we simply use the "table" object instead. For the gripper (our solution is based on the Panda Hand model provided here: https://github.com/JenniferBuehler/graspit-pkgs/issues/55#issue-515423230, Accessed: 12 December 2022), we defined the virtual contacts using the user interface of Graspit! and modified the inventor files to include appropriate scaling so the size of the gripper in Graspit! matches the real-life gripper. We ran the Graspit! grasp planning a total of 3 times for pick grasps and 5 times for place grasps, from which we acquired 23–26 pick grasps, and around 30 place grasps for each frame after the initial filtering (removing multiples of pick grasps and place grasps or grasps with high contact energy). Based on geometric proximity, 5 of the pick grasps were classified as valid, and the rest were classified as invalid (see Figure 2).

The fine-tuning dataset was generated using the first 20 frames (the object pose is randomized for each frame) from Blender. For these, a total of 489 pick grasps were classified as either valid or invalid. During converting the data into Dex-Net 2.0 format, we added the additional 12 invalid grasps for each of the 489 original ones to enforce precise gripper orientation prediction. This resulted in a total of 6357 grasps, of which the ratio of grasps with positive labels was around 1.57%.

On this dataset, we fine-tuned the Dex-Net 2.0 pre-trained GQCNN model, for 60 epochs using a batch size of 64. This fine-tuned model is evaluated against the original Dex-Net 2.0 pre-trained model. The success rate of the pick-place operations was evaluated by comparing the original and the fine-tuned GQCNN. For this evaluation, we used 20 additional frames (not used for training or validation) that use the same camera placement as in the setup for generating the training dataset, and 40 other frames generated using a camera placement different from the training setup. The findings of these evaluations are reported separately to highlight the method's sensitivity to the variation of the setup.

3.2. Results Using Camera Extrinsic Parameters from Training Setup

Table 1 shows the results of our evaluation using synthetic depth images from Blender and the same camera extrinsics as in the setup for generating the training data. As a result, in these evaluation frames, only the object pose was randomized. It can be seen that our fine-tuned GQCNN outperformed the original Dex-Net 2.0 pre-trained model significantly.

Table 1. Evaluation of our fine-tuned GQCNN against the Dex-Net 2.0 pre-trained GQCNN for depth images generated from a camera with the same extrinsic parameters as in the training setup.

Model	Success Rate (Task)	Success Rate (Pick)	Success Rate (Place)
Dex-Net 2.0 GQCNN	0.65	1.0	0.65
Fine-tuned GQCNN	**0.9**	1.0	0.9

It is important to note that none of the predicted grasps from either model resulted in a collision during picking the object. All the failed grasps in our experimental setup can be attributed to the fact that the models failed to predict an appropriate grasp in the given task context. Thus, the predicted grasp resulted in a collision during placing the object. Having a higher success rate for picking the object than placing it is to be expected since out of all the possible grasps for picking the object, only a few will be feasible for placing it as well. As a result, the probability of predicting a grasp that will be successful for placing the object too is lower. The results suggest that fine-tuning a GQCNN on a small number of synthetic grasps labeled with the task-specifics in mind can significantly increase the probability of the GQCNN model predicting a grasp which will be successful for the whole task (both picking and placing the object).

Given the assembly information (assembly order, objects' relative poses), scene setup, and the 3D models of the objects and the robot, our method can quickly be applied to new objects or new assembly tasks. As a result, the grasp prediction pipeline can be adjusted flexibly for novel scenarios, even within a single day, including scene setup in Blender, fine-tuning dataset generation, and GQCNN model training (especially since the scene, robot and camera setups usually do not change as frequently as the assembly tasks, so they only need to be created once).

3.3. Results Using Different Camera Extrinsic Parameters

Table 2 demonstrates the results obtained by evaluating our fine-tuned GQCNN model against the Dex-Net 2.0 pre-trained GQCNN on 40 synthetic depth images, which were prepared with a camera location different from the one used in the setup for generating the fine-tuning dataset. As can be seen, the difference between the performance of the two models is not as apparent as it was when using camera extrinsics from the training setup. This suggests that the fine-tuned model's predictions for setups different from the training setup align more with the predictions of the original GQCNN. This shortcoming could potentially be mitigated by automatically varying the camera pose in Blender for the training dataset generation. It is a promising way for future improvement to make the fine-tuned GQCNN models robust to changes in camera setup.

Table 2. Evaluation of our fine-tuned GQCNN against the Dex-Net 2.0 pre-trained GQCNN for depth images generated from a camera with the different extrinsic parameters from the training setup.

Model	Success Rate (Task)	Success Rate (Pick)	Success Rate (Place)
Dex-Net 2.0 GQCNN	0.5	1.0	0.5
Fine-tuned GQCNN	**0.55**	1.0	0.55

4. Discussion

In this paper, we proposed an automated pipeline for synthetic training data generation and fine-tuning of GQCNN models for task-specific grasp planning in robotic assembly scenarios. Our method assumes that the object models and assembly information, such as assembly order and the relative pose of objects in the assembly, are known in advance. We generated robust grasps for picking the object using Graspit! and automatically classified these grasps into valid and invalid ones based on their similarity to grasps from another set, generated from Graspit! for the placing of the object. For the set of valid and invalid grasps, we automatically generated synthetic depth images using Blender, and we used these synthetic images to fine-tune a GQCNN model. As a result, the fine-tuned GQCNN model learned to predict grasps with a high probability of success, which is feasible for picking the object and placing it into the assembly as well.

We evaluated our method on a simple simulated scenario and showed that a fine-tuned GQCNN can significantly outperform the original Dex-Net 2.0 pre-trained model, in the context of grasp planning for a specific task. Our method can quickly be applied to train GQCNN models for a flexible robotic assembly scenario.

One limitation of the proposed method is that in Graspit!, only the robot gripper is included during determining the pick and place grasps. While this may work well for simple scenes of tabletop assembly, it may result in a fine-tuned GQCNN, which predicts infeasible grasps in a highly cluttered scene due to the robot's self-collisions or interferences with the environment during task execution. A possible future solution would be to use MTC for determining valid and invalid grasps, where collision for the whole robot arm could be considered instead of the currently used proximity-based method. During our experiments, we concluded that training data generated with only a single camera pose results in a fine-tuned GQCNN that loses its edge against the original pre-trained model in scenarios when the camera pose is different from the one used for the generation of the training data. In the future, the robustness of fine-tuned GQCNNs could be examined by including randomization in the camera pose for generating the training data. Additionally, this paper focuses on the geometric evaluation of predicted grasp poses (via collision detection), but an evaluation in a dynamic environment could also be performed in the future.

Supplementary Materials: The supporting information can be downloaded at: https://github.com/ABC-iRobotics/Task-Specific-Grasp-Planning-GQCNN.

Author Contributions: Conceptualization, A.I.K. and P.G.; methodology, A.I.K.; software, A.I.K.; validation, A.I.K.; writing—original draft preparation, A.I.K.; writing—review and editing, A.I.K. and P.G.; visualization, A.I.K.; supervision, P.G.; project administration, P.G.; funding acquisition, P.G. All authors have read and agreed to the published version of the manuscript.

Funding: The authors thankfully acknowledge the financial support of this work by the project no. 2019-1.3.1-KK-2019-00007 implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2019-1.3.1-KK funding scheme. Péter Galambos is a Bolyai Fellow of the Hungarian Academy of Sciences. Péter Galambos is supported by the UNKP-22-5 (Bolyai+) New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets were analyzed in this study. These data can be found here: https://github.com/ABC-iRobotics/Task-Specific-Grasp-Planning-GQCNN, accessed on 12 December 2022.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

GQCNN	Grasp Quality Convolutional Neural Network
ROS	Robot Operating System
BAT	Blender Annotation Tool
MTC	Moveit! Task Constructor

References

1. Miller, A.T.; Allen, P.K. Graspit! a versatile simulator for robotic grasping. *IEEE Robot. Autom. Mag.* **2004**, *11*, 110–122.
2. Levine, S.; Pastor, P.; Krizhevsky, A.; Ibarz, J.; Quillen, D. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *Int. J. Robot. Res.* **2018**, *37*, 421–436.
3. Balasubramanian, R.; Xu, L.; Brook, P.D.; Smith, J.R.; Matsuoka, Y. Physical human interactive guidance: Identifying grasping principles from human-planned grasps. *IEEE Trans. Robot.* **2012**, *28*, 899–910.
4. Roa, M.A.; Suárez, R. Grasp quality measures: Review and performance. *Auton. Robot.* **2015**, *38*, 65–88.
5. Mahler, J.; Liang, J.; Niyaz, S.; Laskey, M.; Doan, R.; Liu, X.; Ojea, J.A.; Goldberg, K. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv* **2017**, arXiv:1703.09312.
6. Goldfeder, C.; Allen, P.K. Data-driven grasping. *Auton. Robot.* **2011**, *31*, 1–20.
7. Goldfeder, C.; Ciocarlie, M.; Dang, H.; Allen, P.K. The columbia grasp database. In Proceedings of the 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 12–17 May 2009; pp. 1710–1716.
8. Mahler, J.; Pokorny, F.T.; Hou, B.; Roderick, M.; Laskey, M.; Aubry, M.; Kohlhoff, K.; Kröger, T.; Kuffner, J.; Goldberg, K. Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; pp. 1957–1964.
9. Costanzo, M.; De Maria, G.; Lettera, G.; Natale, C. Can robots refill a supermarket shelf? Motion planning and grasp control. *IEEE Robot. Autom. Mag.* **2021**, *28*, 61–73.
10. Costanzo, M. Control of robotic object pivoting based on tactile sensing. *Mechatronics* **2021**, *76*, 102545.
11. Kokic, M.; Stork, J.A.; Haustein, J.A.; Kragic, D. Affordance detection for task-specific grasping using deep learning. In Proceedings of the 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids), Birmingham, UK, 15–17 November 2017; pp. 91–98.
12. Nikandrova, E.; Kyrki, V. Category-based task specific grasping. *Robot. Auton. Syst.* **2015**, *70*, 25–35.
13. Dang, H.; Allen, P.K. Semantic grasping: Planning task-specific stable robotic grasps. *Auton. Robot.* **2014**, *37*, 301–316.
14. Shin, H.C.; Roth, H.R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; Summers, R.M. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. *IEEE Trans. Med. Imaging* **2016**, *35*, 1285–1298.
15. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3, p. 5.
16. Blender Online Community. *Blender—A 3D Modelling and Rendering Package*; Blender Foundation, Stichting Blender Foundation: Amsterdam, The Netherlands, 2018.
17. Károly, A.I.; Galambos, P. Automated Dataset Generation with Blender for Deep Learning-based Object Segmentation. In Proceedings of the 2022 IEEE 20th Jubilee World Symposium on Applied Machine Intelligence and Informatics (SAMI), Poprad, Slovakia, 2–5 March 2022; pp. 000329–000334.
18. Görner, M.; Haschke, R.; Ritter, H.; Zhang, J. MoveIt! Task Constructor for Task-Level Motion Planning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019.