Grasp for Stacking via Deep Reinforcement Learning

Junhao Zhang, Wei Zhang, Ran Song, Lin Ma and Yibin Li

Abstract—Integrated robotic arm system should contain both grasp and place actions. However, most grasping methods focus more on how to grasp objects, while ignoring the placement of the grasped objects, which limits their applications in various industrial environments. In this research, we propose a model-free deep Q-learning method to learn the grasping-stacking strategy end-to-end from scratch. Our method maps the images to the actions of the robotic arm through two deep networks: the grasping network (GNet) using the observation of the desk and the pile to infer the gripper’s position and orientation for grasping, and the stacking network (SNet) using the observation of the platform to infer the optimal location when placing the grasped object. To make a long-range planning, the two observations are integrated in the grasping for stacking network (GSN). We evaluate the proposed GSN on a grasping-stacking task in both simulated and real-world scenarios.

I. INTRODUCTION

Over the past several decades, robotic grasping has demonstrated a high precision of operation in some applications with tightly controlled conditions, such as car assembling and welding. However, in many tasks, the robotic systems have to handle scenes containing a variety of unpredictable objects. For instance, a messy table could make most existing robotic grasping systems fail [1]. Even though the grasp is successful in such a messy scene, presupposed placing may lead to collision between different shapes of objects. So the growing e-commerce businesses are keen to have a system which can be used in various stacking tasks in warehouses [2].

Most existing approaches for manipulating objects by robotic arms are just to grasp them. Some methods [3]–[5] scan the geometries of objects and grasp them by indexing in the databases. Recently, data-driven methods based on deep neural networks [6]–[8] or deep reinforcement learning [9]–[11] become very popular since they can learn a reliable policy end-to-end with raw image inputs. Again, most of these work focused only on grasping while there are a few exceptions [12]–[14]. Zeng et al. [12] employed pushing as an auxiliary action to disorder tightly packed blocks which offers space for grasping. Gualtieri et al. [13] proposed a method combining the grasping and the placing of a single object (such as erecting a cup) at a fixed position. Jiang et al. made a pioneering attempt to investigate the placing problem [14]. Given an object, they separated it from grasping, which was unable to choose the object needed by the placing area.

To take a step forward, we intend to handle the grasping-stacking task with multiple objects in an end-to-end manner. This task is much harder than the above ones, as three questions have to be answered: 1) Which to grasp? 2) How to grasp? 3) Where to place? Similar to the game of 'Tetris', the goal is to place the boxes tightly on a platform and make the stack form solid horizontal lines without gaps.

To accomplish the grasping-stacking task as shown in Fig. 1, we present an autonomous grasping for stacking network (GSN) consisting of two modules, namely the grasping network (GNet) and the stacking network (SNet), which can predict the pixel-wise grasping position and the block-wise stacking position respectively. In order to transfer the information of stacking area to GNet, the features of the heap image (corresponding to the objects in the stack) are fused with the features of the desk image (corresponding to the objects to be grasped). Hence, the GNet considers not only which object is easy to grasp, but also which one is currently needed in the stacking area.

Furthermore, in order to accelerate the learning by forcing the networks to focus on the perceptual information relevant to the task and provide additional training signals, we introduce three auxiliary tasks. First, the desk object number prediction task, using the features extracted from the perception layers of GNet to predict how many objects are left on the table. Second, the heap height prediction task, using the information yielded from the perception layers of SNet to predict the pixel-wise stacked object height. This task helps the network extracting outline features of the heap, which contains useful information to evaluate the current state. Finally, the object-centric feature learning task, to ensure the object which disappears from the desk is similar to the one that is added to the pile, in feature level. That is, the object features captured from different viewpoints (desk

Fig. 1. Example of the grasping-stacking task. The proposed method first selects a box based on the observations of the desk and the pile, and then grasps it and places it in a suitable position.
image and heap image) should be close.

The whole grasping-stacking process is formulated as a Q-learning problem. We employ the distribute prioritized experience replay [15] to learn policies that could grasp different size of boxes and place them tightly on a platform. Experiments are conducted in both simulation and real world to validate the effectiveness of the proposed method.

II. RELATED WORK

Early methods for robotic grasping were based mainly on model-based reasoning which modeled not only the object shape [16]–[18], but also the contact forces and their resistances [19], [20]. Using these methods to control real robotic arms typically requests the computation of grasps from the datasets of object models and the estimation of the object poses by indexing at run time. The performance of the methods depends on the qualities of model fitting and surface reconstruction. Some recent data-driven methods based on deep neural networks [21]–[23] or deep reinforcement learning [9], [24], [25] achieved better grasping performance. Compared with the model-based methods, they relied on convolutional networks to extract features and learned useful information from images without using the specific knowledge of objects.

One unsolved issue raised by robotic grasping is what to do next. The majority of the grasping techniques focused only on grasping objects [26]–[28] while ignoring the purpose of the grasping (e.g., placing an object at another location). To address this issue efficiently, [29] jointly learned both the task-oriented grasping of a tool (e.g., a hammer) and the manipulation policy of the tool (e.g., using a hammer to sweep or hammer). In [13], [30], grasping was trained for facilitating robots to put objects with the right orientation (e.g., place a cup upright) at a fixed position, which also needed to estimate suitable positions to grasp the objects by some specifically designed grasp descriptors.

The work most relevant to ours is [14] where Jiang et al. proposed a method for grasping objects and placing them in different areas. But our method is fundamentally different since 1) unlike Jiang’s method, we train the grasping and stacking policies jointly rather than separately, 2) we use a deep network rather than a handcrafted method to learn important perceptual features of the objects directly from the raw images, and 3) our deep reinforcement learning framework can learn a strategy from scratch by self-supervised trial and optimization without any costly manual effort for annotation. We believe that this is the first system to perform model-free reinforcement learning with deep networks, which grasps objects for appropriately placing them by learning actions from visual observations in an end-to-end manner.

III. METHOD

In this section, we elaborate our system including the Q-learning formulations, the network architectures and the training protocols.

A. Problem Formulation

The robotic manipulation can be expressed as a Markov Decision Process (MDP). As shown in Fig. 2, at the state \( s_t \) where \( t \) denotes a time step, the system captures the placing and the grasping areas through two cameras respectively. Then the robot selects and implements an action (consisting of grasping and placing an object) \( a_t \) subject to a policy \( \pi_\theta(s_t) \) parameterized by \( \theta \) which can be learned through training a deep network. And then the state updates to \( s_{t+1} \) with an immediate reward \( r(s_t, a_t) \). After training, this reinforcement learning problem can be solved by finding an optimal policy \( \pi_\theta^* \) which maximizes the expected sum of the future reward with a discount factor \( \gamma \). Thus the objective function \( F \) can be defined as:

\[
F = \max_{\theta} \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[ \sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t) \right].
\]  

The framework above provides a solution to such decision-making problems, but learning it is challenging due to the difficulty of data collection. As shown in [9], [11], collecting a large amount of experiences is crucial to the performance of reinforcement learning networks. Compared to on-policy learning, off-policy methods can reuse the collected data to enable the training process when data collection is difficult.

To efficiently train the network, inspired by [11], we implement the off-policy Q-learning algorithm [31] to learn a policy estimating the Q-function by minimizing the Bellman error:

\[
\varepsilon = \frac{1}{2} \mathbb{E}_{s_t, a_t \sim \pi_\theta} \left[ (Q_\theta(s_t, a_t) - r(s_t, a_t) + \gamma \max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}))^2 \right].
\]  

After training, this policy will choose actions by maximizing the optimal action-value function \( Q_\theta^*(s_t, a_t) \), forming an optimal policy \( \pi_\theta^* \). In other words, it will choose the action \( a_t \) at the state \( s_t \) which yields the maximum cumulative rewards, leading to grasping the object which is needed in the stacking area and putting it in a suitable place in current and future steps.

B. State Representations of Grasping and Stacking

Similar to [12], [32], we model the grasping states \( s_{gt} \) by 4-channel RGBD images captured in front of the desk. Before feeding them into the networks, the 3-channel color data combined with depth data are orthographically project to the overlook viewpoint and rotate different angles \( \theta \in \{0, 22.5, 45, 67.5, 90\} \). This strategy generates eight new color heightmaps as shown in Fig. 2. For the stacking state representation, we use RGB images taken by a camera facing the placing area. Since the gripper cannot operate when the fingers being blocked by other objects, we designed the stacking position as a single layer arrangement (extends along the \( x \)- and \( z \)-axes while keeps fixed along the \( y \)-axis) as shown in Fig. 2, so the robotic arm could place objects along the \( y \)-axis with no collision. Therefore, the placement of objects, state \( s_{st} \) can be fully depicted by 2-dimension RGB pictures.
C. Actions of Grasping and Stacking

We define the representation of a grasping action as $a_{gt}$ and a stacking action as $a_{st}$, which generates an action $a_t$ at each time step. As for a grasping action $a_{gt}$, it composes Cartesian motion command $[x_g, y_g, z_g, \theta_g]$, where $[x_g, y_g, z_g]$ corresponds to the center of the gripper when grasping and $\theta_g$ is a rotation of the wrist around the $z$-axis (vertical axis). This method is similar to the pixel-wise grasping system proposed by Zeng et al. [12] except that we use $180^\circ$ to divide it into 8 separate $\theta_g$ rotations instead of $360^\circ$ divide by 16, because of the symmetry of objects. As for stacking action $a_{st}$, we separate stacking area to 14 positions along $x$-axis, denoted by $s_1 \in [0, 13]$. Since our system grasps at object center after training, those areas also represent the center of placed objects. The most left as well as most right blocks didn’t include in action space since most of boxes in our tasks are of 3 block width (if placed in the edge, a part of box will out of sight). Note that the robotic arm places objects by not only $x$ coordinate specified by $f_x(s_i)$ ($f_x$ is a discrete function mapping $s_i$ to $x$ coordinate) but also $z$ coordinate inferred by $s_{zt}$. Other commands such as $y$ coordinate and gripper orientation are fixed in stacking actions, which simplify this stacking problem and help placing boxes in a dense form.

D. Rewards of Grasping and Stacking

Our reward scheme for reinforcement learning contains two parts with one for the stacking metrics and the other for the grasping decision. Since the well stacked boxes are tight and level on the top, we define the stacking reward $r_s$ as below:

$$r_s = B^\downarrow - H^\uparrow - O^\downarrow - L,$$

where $B^\downarrow$ denotes the bumpiness decreased value (calculated by the variance of the column-wise heights thresholded to -0.3, 0 and 1). $H^\uparrow$ denotes the maximum height increased value (0 or 0.7 by thresholding). $O^\downarrow$ denotes the number of holes. Once a gap is covered, a hole forms and can not be filled any more. $L$ is a binary value representing if the top of stack is level (0) or not (1). These four values are calculated by comparing $s_{pt+1}$ with $s_{pt}$. The former three are different piecewise functions whose inputs are images $s_{pt+1}$ and $s_{pt}$, guiding the policy to implement a stacking with a high fitness. Inspired by the early termination strategy proposed by Peng et al. [33], the metric $L$ takes effect if the learned policy fails to implement a stable stacking, and sends a restarting signal since the following rewards would be inaccurate. The grasping reward $r_g$ is defined by:

$$r_g = G - D + \frac{r_s}{4},$$

where $G$ denotes the grasp result with 0 representing a failed grasp and 1 representing a successful one. $D$, denoting the distance between the object center and the grasping position, is crucial for achieving a high stacking precision. The reason why $r_s$ is added to $r_g$ will be explained in Section. III-F.

E. Network Architecture

We extend the deep Q network [34] by coalescing different features in a fully convolutional network (GNet). As shown in Fig. 2, we model two Q-functions by two convolutional networks (GNet and SNet). At each time step, GNet evaluates the grasping Q-function of every pixel in $s_{gt}$, and SNet evaluates the stacking Q-function of each block in $s_{st}$. 

Fig. 2. Illustration of the proposed GSN. The static camera set in front of the table captures the area within the blue dotted lines. The wrist camera carried by the robotic arm moves toward the stack and takes an image of the area within the red dotted lines. Note that the column-wise height prediction is shown as colorful dots in the visualized image of $Q_g$. The predicted Q values in $Q_g$ and $Q_s$ are visualized as heat-maps where the warm colors correspond to the positions suitable for stacking and grasping respectively. In the testing, we use the actions corresponding to highest Q value, marked with the white circle.
Similar to [35], both networks in our structure use the first 3 blocks of ResNet-50 [36] to extract the features from the raw image data. Since the depth information is important for a precise grasp, we adjust the input layers of GNet from 3 channels to 6 channels (i.e. RGB to RGBDD by concatenating the RGB channels and the channel-wise cloned depth channels). The input channels of the ResNet component in SNet remains unchanged. We resize both the grasping observation (with the resolution of 224x224) represented by $s_g$ and the placing observation (with the resolution of 256x128) represented by $s_{pt}$ to 320x320, which generates suitable sizes of feature maps later used for the upsampling and the auxiliary tasks.

For the following convolutional layers in GNet and SNet, they share the same network architecture as illustrated in Fig. 2. Every convolutional layer’s kernel size is 1x1 as suggested by Lin et al. [37], which facilitates dimension reduction and mitigates possible overfitting. To combine the information with regard to which object is required in the stack to form an ordered layout and which one is easy to grasp in a particular grasping orientation, we fuse the high-level features of $s_{pt}$ generated by the convolutional layers $\Phi_s$ in SNet with the low-level features of $s_{pt}$ generated by the convolutional layers $\Phi_g$ in GNet. Considering that GNet is a fully convolutional network which encodes positional information in feature maps, instead of concatenating $\Phi_s$ with $\Phi_g$ at the channel level, we adopt two linear layers to transfer $\Phi_s$ into a channel-wise weight $\omega_g$, and then multiply $\omega_g$ (between 0 and 1) with $\Phi_g$ as follows:

$$\Phi_m = \lambda \omega_g \Phi_g + (1 - \lambda) \Phi_g,$$

where $\Phi_m$ denotes the fused features and $\lambda$ is a scaling factor which balances the original features (containing preliminary predictions of which position is easy for grasping the object) and the fused features (highlighting the features extracted from $s_{pt}$ of which is useful for a suitable placement). We set $\lambda = 0.25$ in our framework.

The aforementioned features are sent to different layers for diverse tasks. As for the GNet, mixed low-level features $\Phi_m$ are processed by two convolutional layers, and then fed into a bilinear upsampling layer. Another feature $\Phi_g$ is also used for predicting the number of object on the desk through an average-pooling layer followed by the ReLU and linear layers, which assists the perception layers to distinguish different objects and be sensitive to the small ones which tend to be neglected. In the SNet, the high-level feature $\Phi_s$ is used for not only facilitating the GNet to make wider perception, but also predicting block-wise Q values and column-wise height values. Two separate linear layer modules are used for estimating Q values $Q_s(s, a_s)$ jointly by calculating the value function $V_s(s)$ and the advantage function $A_s(a)$. This is analogous to the dualing networks architecture in [38]. For the height prediction task, the objects will form a stack if they are placed one by one, and the outline of this stack contains implicit information to predict Q values for the stacking.

Another auxiliary task which is not shown in Fig. 2 is the object-centric feature learning task proposed by Jang et al. [35]. Different from Jang’s method, we extend the concept of object persistence to two parts: when a robot grasps an object from the desk and place it, the representation of both scenes (desk image and heap image in Fig. 1) should change according to the features of the object that was removed form desk and also appear on the stack. This task compares low-level features $\Phi_g$ with $\Phi_p$ before and after a successful grasping followed by the stacking action sequence, assuming that the same object should have similar feature vectors in both the desk (estimated by $\Phi_g(t) - \Phi_g(t + 1)$) and the stack (estimated by $\Phi_p(t + 1) - \Phi_p(t)$). To calculate these features, we process the feature maps $\Phi_g$ and $\Phi_p$ by applying a global average-pooling and ReLU non-linearity layer as suggested by [35]. Thus our perception modules in GNet and SNet have the ability to recognize the same object through similar features, which assists the aforementioned feature fusion.

F. Learning State-Action Value Functions

We jointly train the GNet and the SNet using the deep Q-networks [34] as the Q-function approximators. Specifically, we model the grasping Q-function $Q_g(s_{pt}, a_g)$ as the fully convolutional network (GNet), and the stacking Q-function $Q_s(s, a_s)$ as the dualing network (SNet).

We use double Q-learning [39], [40] to train both GNet and SNet. This method is more reliable compared to the single Q-learning method by employing target networks and modified max operator. The target networks share the same architecture as networks in Fig. 2 (without the auxiliary task modules), whose parameters are copied every 300 steps from the online models. For the max operator in Equation 1, the double Q-learning uses the action that maximizes the current $Q_{\theta}$, and the value obtained from the target $Q_{\theta-}$, leading to the following loss function for both the grasping and the stacking Q-functions:

$$\ell_i = \frac{1}{2} \mathbb{E}_{s_{it}, a_{it} \sim \pi_{\theta, \lambda}} \left[ Q_{\theta}(s_{it}, a_{it}) - \max_{a_{it+1}} Q_{\theta^-}(s_{it+1}, a_{it+1}) \right]^2,$$

where $i \in \{g, s\}$ in each training epoch. Note that $\ell_s$ is computed only when the grasping is successful, which leads to the change of $s_{pt}$. The expectation $\mathbb{E}$ is calculated by batch samples.

As described in [12], [32], a fully convolutional network used for pixel-wise parameterization of both state and action space provides efficient computation, and the parameterization of gripper locations (pixel-wise sampling) and orientations (by rotating $s_{pt}$) enables the convolutional features to be shared across locations and orientations. In our work, the GNet predicts the Q value $Q_g$ representing the grasping at the location that is likely to succeed and the object that is needed by the stacking policy. Therefore, the reward for the Q-function $Q_g(s_{pt}, a_g)$ should contain both the grasping and the stacking rewards. By this way, the feature fusion layers which transfer $\Phi_s$ to $\omega_s$ are capable of adjusting itself during training.

To accelerate the training, we implement distributed learning framework proposed by Horgan et al. [15] with 16
Algorithm 1 Grasping for stacking policy learning
1: \( \theta_0 \leftarrow \text{Initialize Network}() \)
2: Initialize \( M_g \) (grasping prioritized replay memory) and \( M_s \) (stacking prioritized replay memory).
3: while \( \min(M_g, M_s, s) < \text{Batch Size} \) do
4: \( \text{Wait a minute (for workers generating experiences).} \)
5: \( t \leftarrow 0 \)
6: \( \text{while } t < \text{Training Rounds} \) do
7: \( \text{id, } (s, a, r, s') \leftarrow M_g, \text{Sample(Batch Size)} \)
8: \( \ell_g, \ell_s, \text{error}_g \leftarrow \text{Compute Loss}(s, a, r, s'; \theta_t) \)
9: \( \theta_{t+1} \leftarrow \text{Update Parameters}(\ell_g + \ell_s, \theta_t) \)
10: \( \text{P} \leftarrow \text{Compute Priorities(error}_g) \)
11: \( M_g, \text{Set Priority(id, P)} \)
12: \( t \leftarrow t+1 \)
13: \( \text{id, } (s, a, r, s') \leftarrow M_s, \text{Sample(Batch Size)} \)
14: \( \ell_g, \ell_s, \text{error}_s \leftarrow \text{Compute Loss}(s, a, r, s'; \theta_t) \)
15: \( \theta_{t+1} \leftarrow \text{Update Parameters}(\ell_g + \ell_s, \theta_t) \)
16: \( \text{P} \leftarrow \text{Compute Priorities(error}_s) \)
17: \( M_s, \text{Set Priority(id, P)} \)
18: \( t \leftarrow t+1 \)
19: \( \text{end while} \)
20: \( \text{end while} \)

workers sampling in an asynchronous order. After collecting 200 transitions, a worker transfers the experience with different priorities to the learner and copies the parameters from it. Meanwhile, the learner implements the training by two prioritized experience replay buffers which store the experience indexes with different priorities related to \( \ell_g \) and \( \ell_s \) and reuse the prioritized experience replay samples alternately. So the samples with greater prediction errors will have higher priorities. The pseudo code for the training routine of the learner is listed in Algorithm 1.

G. Training Details

We train our system in a simulated environment for efficiency. We use a UR5 robotic arm equipped with a Robotiq85 gripper in V-REP [41] (see Fig. 3). We design four types of boxes with the sizes of 1x1x1, 1x3x1, 2x3x1 and 3x3x1 blocks respectively. To accelerate the sampling process, we stack objects by directly changing the position of them (so that the objects are always within the camera view as mentioned in Section III-C).

Apart from the losses for estimating the Q-function, auxiliary tasks have different losses. The tasks of predicting the metrics composed of the bumpiness and the maximum height defined in Section III-D. We randomly add \( n \in [6, 9] \) objects in the training and testing scenes.

We train the GNet and the SNet simultaneously using stochastic gradient descent with a fixed learning rate of 0.0001. Both Q-learning methods adopt the \( \epsilon \)-greedy exploration strategy with \( \epsilon \) initialized at 0.9 for the SNet and 0.5 for the GNet respectively, and then annealed over training to 0.05. The discounted factor \( \gamma \) in Equation 6 is set to 0.5.

![Fig. 3. Example of the simulated environment. Two cameras are placed statically, and displayed as blue cuboids with radial lines. The images captured by the cameras are shown at the top right.](image)

IV. EXPERIMENTAL RESULTS

We evaluate our system (GSN) in both simulated and real-world scenes. We randomly place boxes in different sizes and colors on a table and the robot needs to grasp and place the boxes one by one to form a stack. We carried out three experiments: 1) a comparative study between our reinforcement learning framework and a supervised learning method; 2) an ablation study for assessing the contributions of each component of our system in terms of overall performance; 3) a demonstration that our system can be applied to a real robot for the grasping and stacking task.

A. Evaluation Metrics

To make a fair comparison, we introduce two kinds of metrics: the grasping metrics is the average grasp distance (G-distance) between the gripper center and the object center; the stacking metrics composed of the bumpiness and the maximum height defined in Section III-D. We randomly add \( n \in [6, 9] \) objects in the training and testing scenes.

B. Comparison with the Baseline Method

Because the system proposed by Jiang et al. [14] was unable to stack objects next to each other, to demonstrate the reinforcement learning method with our GSN architecture is suitable for the grasping-for-stacking task, we designed a supervised learning method named as Supervised GSN (SGSN) with a similar architecture as ours.

More specifically, the Supervised GSN is a policy that has the same state and action space as ours described in Section III. The major difference between the Supervised GSN and the proposed GSN is the annotations. The GNet in the SGSN is trained through a supervised learning with the annotation of binary classification obtained from trails and error estimation, inferring the pixel-wise grasping affordance value between 0 and 1. The SNet in SGSN is also changed in an analogical way, which learns with binary annotations.
where 1 represents the best stacking block in the action space and other stacking blocks are labeled with 0. This position is defined by the lowest top area of stack (including bottoms if this block is empty) which could contain a wide box of 3 blocks at least. Since this supervising signal for grasping policy is incapable of optimizing the fusion layers, we remove it from the network.

Table. 1 shows the performance of the SGSN and our GSN. In terms of grasping (G-distance), GSN has a smaller grasping area than SGSN, which means the gripper is more close to the center of the target object in GSN. In terms of stacking metrics (bumpiness and max height), our system outperforms SGSN due to the lack of information fusion in the SGSN. As presented in Fig. 4, GSN offers a better solution for stacking objects of various sizes (especially the widths), which benefits from the stack-task driven grasping. Guided by artificial annotations for stacking, SGSN prefers to grasp and stack big boxes, forming uneven staking such as Episode 20, 120 in Fig. 4.

C. Ablation Study

To show the benefit of the proposed GSN for the grasping and stacking task, we carried out an ablation study by comparing the GSN with the GSN-no-fus (trained without the fusion layers). We also implemented an ablation study for the auxiliary tasks where we measured the performance for 3 variants: GSN-no-height (trained without the height prediction), GSN-no-num (trained without the prediction of object number) and GSN-no-fl (trained without the object-centric feature learning task).

Fig. 4 shows the learning curves corresponding to the ablated versions of the GSN. As expected, the GSN outperforms the GSN-no-fus. The possible reason is that the grasping decision is made without the information of the stacking area in the GSN-no-fus, which makes the grasping and the stacking policies work separately and thus less optimally. In particular, the GSN-no-fus failed to place the 1x1 box at the best position since its width is different from others. The comparative results also suggest that the auxiliary tasks are useful for accelerating the learning speed (especially with the object number prediction task), and the height prediction task as well as the object-centric feature learning task also improves the final performance.

D. Real World Performance Evaluation

We evaluate the same testing task on a real robot, using a UR5 robotic arm and a Robotiq85 gripper with two Realsense camera mounted on a tripod to capture the RGBD images of the grasping area and the wrist of the robot to take RGB images of the placing area.

In the real world test, the stacking performance is evaluated by the altitude difference $H_d$ between the highest and the lowest surfaces of the stack. If $H_d \leq 2$, the stacking task is considered as a successful one. Our method performs 75%(15/20) stacking success rate in the box stacking task, and SGSN only achieves 15%(3/20) success rate since it can hardly stack after inaccurate grasp. The video of testing in the real world environment is available at https://vsislab.github.io/gsdrl/.

V. Conclusions

In this work, we propose a system which jointly learns the grasping and the stacking policies through the grasping-for-stacking network (GSN) that enables a robotic arm to pick boxes from a table and put it on a platform properly. We demonstrate that the use of the fusion layers between the grasping network and the stacking network can integrate the information of both areas, leading to a desired deep reinforcement learning which enables the grasping policy to make decisions with the consideration of the stacking as well.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under Grants U1913204 and 61991411, in part by the National Key Research and Development Plan of China under Grant 2017YFB1300205, in part by the Shandong Major Scientific and Technological Innovation Project (MSTIP) under Grant 2018CXGC1503, and in part by Taishan Young Scholarship No.tsqn201909029.