2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

An offline learning of behavior correction policy for vision-based robotic manipulation

Qingxiuxiong Dong[1], Toshimitsu Kaneko[1] and Masashi Sugiyama[2,3]

***Abstract*— Offline learning usually requires a large dataset for training. In this paper, we focus on vision-based robotic manipulation tasks and utilize certain task properties to achieve offline learning with a small dataset. We propose a two-stage agent consisting of a tentative decision stage and a correction stage, where the tentative decision stage determines a tentative action from the original camera image, and the correction stage determines a correction to the tentative action based on the cropped image according to the tentative action. The correction stage utilizes task properties to obtain the cropped image with task-relevant features, enabling efficient correction. In particular, the training of the two stages can be performed individually, which enables a straightforward application of general offline learning algorithms. We conduct experiments by combining the two-stage agent with conventional offline reinforcement learning and imitation learning algorithms. In both cases, we benchmark the proposed method using RLBench and demonstrate that the task performance is significantly improved by the correction stage.**

I. INTRODUCTION

Reinforcement learning (RL) is a major tool for learning robotic manipulations. Recent advances in deep learning have enabled the learning of vision-based control in an end-to-end manner [1], [2], [3], [4]. Conventional online RL algorithms require trial and error to obtain new data, and a large amount of interaction with the environment is usually required. Moreover, there are safety concerns when applied to real robots such as damaging the robot, indicating that substantial progress is required for its practical application. Offline RL can learn from existing datasets without any safety concerns, but current offline RL algorithms usually require a large amount of data to train an agent. In this paper, we focus on the offline method for vision-based robotic manipulations, and explicitly utilize the properties of the robotic manipulation tasks to achieve efficient usage of datasets.

A notable property of robotic manipulation tasks is that the interaction between the robot and the objects being manipulated occurs near the end-effector. Compared to using the whole image, we can obtain more task-relevant information by focusing on the pixels near the end-effector and improve the accuracy of the control [5], [6], [7], [8], [9], [10]. For example, [5], [6] presented an algorithm for grasping tasks by gradually zooming into and cropping the pixels where grasping is expected to be performed. The cropped images provide more task-relevant information and improve the success rate of the tasks. However, training such an agent is not straightforward especially in an offline setting, as this algorithm requires a training on the process of the cropping, which is usually not included in the offline datasets.

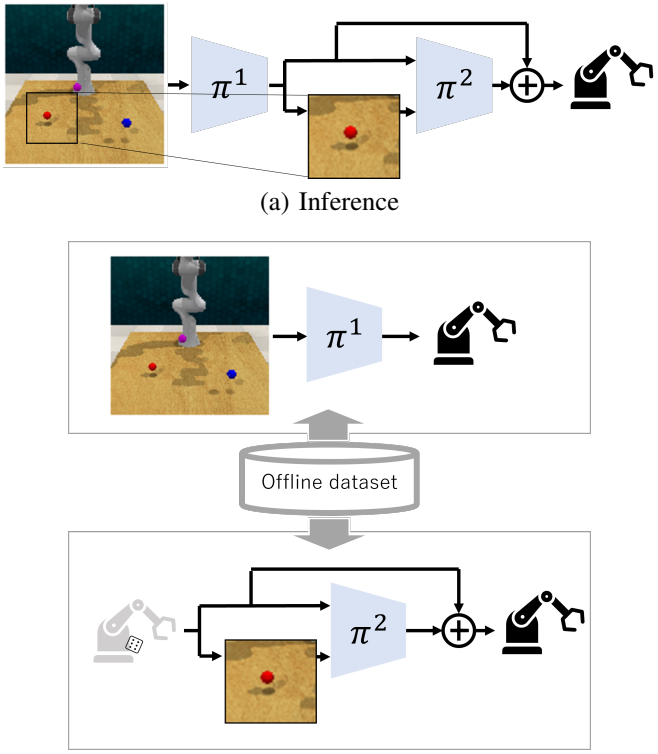

Fig. 1. Illustration of our approach. (a) We determine the action (including the next end-effector pose) in two stages. The first stage determines a tentative action. The second stage crops the image base on the tentative action and determines a correction action. (b) The training of the two stages can be independent, and it is straightforward to apply conventional offline RL and imitation learning.

In this paper, we propose a two-stage agent consisting of a tentative decision stage and a correction stage, and we show that our approach enables an effective utilization of task-relevant information in the offline setting, which was not achievable in the previous works. The first stage, the tentative decision stage, is a conventional agent that determines an action based on the input state, including the camera image. The second stage, the correction stage, crops the camera image based on the tentative action and determines a correction action to the tentative action. The objective of the second stage is to correct the action determined by the first stage, using more task-relevant information from the cropped image. In particular, the two stages can be defined

[1]Media AI Laboratory, Corporate Research and Development Center, Toshiba Corporation, Kawasaki, Japan.
[2]Center for Advanced Intelligence Project, RIKEN, Japan.
[3]Graduate School of Frontier Sciences, The University of Tokyo, Japan.

independently as follows: The first stage merely determines an action as a conventional agent, independent of the second stage. The second stage performs corrections to arbitrary actions, irrespective of the first stage. This is equivalent to assuming that the second stage works when the first stage is replaced by a policy that may output arbitrary actions, and this replacement can also be regarded as data augmentation. The independence of the two stages allows them to be trained individually, and it is straightforward to apply common offline algorithms to each of them for training. Note that [11] also learns a correction policy, but it differs in that the correction policy depends on the tentative policy, and the task-relevant information specific to manipulation tasks is not utilized. Our approach results in a more efficient learning due to these differences.

We evaluate our algorithm using 8 tasks from RLBench [12] that vary in difficulty. We use 100 trajectories of demonstrations as an offline dataset, and we train the two-stage agent using Behavior Cloning (BC) or Conservative Q-learning (CQL) [13]. Our results demonstrate that the performance of the agent is improved by the correction of the second stage in both cases. In addition, while it is also possible to determine the final action directly instead of the correction in the second stage, our results show that it is better to determine the correction in the second stage.

The main contributions are as follows. We introduce a two-stage agent where the first stage determines a tentative action, and the second stage corrects the tentative action to improve the performance. Each agent can be trained individually, and allows the applications of common offline RL and imitation learning. We perform experiments with RLBench [12] and show that the performance is significantly improved by the correction of the second stage.

II. RELATED WORKS

Cropping in manipulation tasks A common approach for tasks with visual input is cropping. In general tasks, cropping is typically used to obtain a robust visual representation by randomly cropping the image for data augmentation [14]. Meanwhile, in the manipulation tasks, cropping can also be used to obtain more task-relevant features, as these features are usually localized and have strong correlation to the end-effector position. That is, robots interact with the world through the end-effector, and in most tasks, task-relevant information can be obtained from the pixels around the current and next end-effector position. For example, if the task is to place an object, it is sufficient to know how the object is held in the end-effector and where to place it, which corresponds to the pixels near the current end-effector position and the next end-effector position.

Cropping the pixels near the end-effector is a natural idea that is used in many previous works. For example, [9] proposed an algorithm that determines a crop position in the first stage and an action (i.e., end-effector pose) in the second stage. [5], [6], [7], [8] proposed algorithms that perform cropping on the input image according to the action multiple times, and gradually zoom into the pixels where grasping is to be performed. However, these algorithms require modifications to the action space, such as adding the crop position as an action, or adding a variable to determine if another zooming is required. Such modifications make it difficult to apply offline learning algorithms because the modified actions are usually not recorded in the offline data. Moreover, these previous works depend on depth images, which are typically of lower quality, and obtaining high-quality depth images is significantly more expensive compared to obtaining high-quality RGB images.

Offline learning In this paper, we train the agent using offline RL and imitation learning [15], [16]. A major problem in offline RL is the overestimation of the value of out-of-distribution actions. In online RL, if the agent overestimates the value of a certain action, this overestimation can be corrected by executing the action in the environment and observing that the value is overestimated. However, this is not possible in offline RL as interaction with the environment is not allowed. A common solution is to estimate the return of a policy in a conservative manner [13], [17], which restricts the output of the policy to the actions in the offline dataset. In this paper, we use Conservative Q-learning (CQL) [13] as one base algorithm for offline RL.

For imitation learning, the simplest method is Behavior Cloning (BC), which mimics the offline dataset by learning a policy that outputs the actions for the states in the dataset with supervised learning. BC does not use the rewards or transitions in the offline dataset, and is usually only applicable when the offline dataset is generated by an expert policy. If the offline dataset includes non-expert data, it is possible to combine RL algorithms with BC to constrain the policy to be close to the dataset [18] or perform weighted behavior cloning [19], [20]. Here, we assume that expert demonstration data is available, and we only consider the vanilla BC for imitation learning. It is known that even if the offline dataset is generated by an expert policy, offline RL may still outperform BC [21]. Thus, we evaluate our method based on both CQL and BC.

III. BACKGROUND

The RL problem is defined by a Markov Decision Process (MDP) $\{\mathcal{S}, \mathcal{A}, p, r\}$. At each time step t, the agent receives a state in the state space $s_t \in \mathcal{S}$ and decides an action from the action space $a_t \in \mathcal{A}$ according to the policy $\pi(a_t|s_t)$. The agent then receives a reward $r(s_t, a_t)$, and the next state transitions to s_{t+1} according to the unknown transition probability $p(s_{t+1}|s_t, a_t)$. We use ρ_π to denote the state-action marginals of the trajectory distribution induced by π and p. The RL objective is to maximize the discounted reward $\mathbb{E}_{s_t,a_t\sim\rho_\pi}[\sum_t \gamma^t r(s_t, a_t)]$, where $\gamma \in [0, 1)$ is the discount factor.

In the offline setting, a fixed dataset $\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}_t$ collected by an unknown behavior policy is given. The goal is to obtain the best possible policy by training only on the fixed dataset $\mathcal{D}$. BC is the simplest method for training from a fixed dataset if the dataset is collected by an expert policy. BC simply learns

the behavior in the dataset, so that the action predicted by the policy is close to the action in the dataset. In particular, we perform BC by minimizing the mean squared error:

$$\mathbb{E}_{s_t,a_t\sim\mathcal{D}}|\pi(s_t)-a_t|^2. \tag{1}$$

In addition to BC, offline RL is another option for learning from a fixed dataset. CQL is an offline RL algorithm that applies a regularizer to prevent the overestimation of Q-values for out-of-distribution actions. CQL trains a Q-function $Q_\theta(s_t,a_t)$ and a policy $\pi_\phi(a_t|s_t)$. We follow the implementation of CQL with soft actor-critic (SAC) [22] as shown in [13]. The policy π_ϕ is improved in the SAC-style by maximizing

$$\mathbb{E}_{s_t\sim\mathcal{D},a\sim\pi_\phi(\cdot|s_t)}[Q_\theta(s_t,a)-\log\pi_\phi(a|s_t)]. \tag{2}$$

The Q-function Q_θ is trained by minimizing

$$\mathbb{E}_{s_t,a_t\sim\mathcal{D}}[(Q_\theta(s_t,a_t)-\mathcal{B}^\pi\bar{Q}(s_t,a_t))^2]+\alpha\mathcal{R}(\theta). \tag{3}$$

The first term is the Temporal Difference (TD) error with the Bellman backup operator $\mathcal{B}^\pi\bar{Q}(s_t,a_t) := r(s_t,a_t)+\gamma\mathbb{E}_{a_{t+1}\sim\pi(\cdot|s_{t+1})}[\bar{Q}(s_{t+1},a_{t+1})]$ applied to a delayed target Q-function $\bar{Q}$. The second term is the CQL regularizer defined by

$$\mathcal{R}(\theta)=\mathbb{E}_{s_t\sim\mathcal{D},a_t\sim\mu(\cdot|s_t)}[Q_\theta(s_t,a_t)]-\mathbb{E}_{s_t,a_t\sim\mathcal{D}}[Q_\theta(s_t,a_t)], \tag{4}$$

which minimizes the Q-values under a distribution $\mu(a_t|s_t)$. In practice, $\mu(a_t|s_t)$ is calculated by sampling actions from the policy $\pi(a_t|s_t)$ as shown in [13].

IV. METHODS

A. Inference of the two-stage policy

We first explain the inference process of the two-stage policy. We denote the first stage, the tentative policy, by π^1 and the tentative action by a_t^1. Similarly, the second stage, the correction policy, by π^2 and the correction action by a_t^2. We define a transformation function f which transforms the state including the cropping of the camera image according to the tentative action a_t^1. We also define a correction function g which corrects the tentative action a_t^1 according to the correction action a_t^2 to obtain the final action a_t that is executed. Given these functions, the policy that determines an action a_t based on a state s_t is straightforward as follows:

1) Calculate the tentative action $a_t^1\sim\pi^1(\cdot|s_t)$,
2) Transform the state s_t to obtain $s_t^2=f(s_t,a_t^1)$,
3) Calculate the correction action $a_t^2\sim\pi^2(\cdot|s_t^2)$,
4) Obtain the final action $a_t=g(a_t^1,a_t^2)$.

In the following, we explain the details of the transformation function f and the correction function g. We assume that the states and the actions are given in the following form, which is a general form for controlling a 6 degree-of-freedom manipulator with a gripper. A state is given by $s_t=(o_t,z_t)$, a composition of the camera image o_t and the end-effector pose z_t in the same form as the action. An action can be decomposed as $a_t=(x_t,r_t,g_t)$, where $x_t\in\mathbb{R}^3$ is the position, $r_t\in\mathbb{R}^3$ is the orientation in the Euler angle, and g_t is the status of the gripper, for example, $g_t\geq 0$ for open and $g_t<0$ for close.

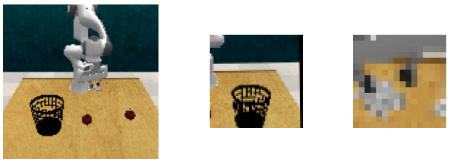

Fig. 2. Example of the input images. (Left) The original camera image o_t, (Middle) The cropped image o_t^2 according to the tentative action, (Right) The cropped image e_t^2 according to current end-effector pose.

Transformation function The primary objective of f is to crop the camera image o_t based on the tentative action a_t^1 to obtain o_t^2, namely, $o_t^2=f_{\mathrm{crop}_1}(o_t,a_t^1)$, where f_{crop_1} denotes a cropping operation as follows. The tentative action a_t^1 consists of the position and orientation of the end-effector. The correspondence between the position of the end-effector and the pixel coordinate can be calculated using calibration data, and we use the position component x_t^1 to obtain the pixel coordinate for cropping. In addition, we use the angle around the optical axis of the camera to obtain the orientation for cropping. Note that one common problem is that task-relevant features may not be located exactly at the end-effector, even if they are close. For example, if we hold one edge of a long object, it is possible that the other edge is also task-relevant. However, we show that our method does not require a small crop size, and this problem can be circumvented by choosing a crop size that ensures sufficient information is included in the cropped image.

While the cropped image o_t^2 provides information about the positions after the action is executed, we also crop the image using the current position of the end-effector to obtain e_t^2, namely, $e_t^2=f_{\mathrm{crop}_2}(o_t,z_t)$ with a cropping operation f_{crop_2}, which provides information such as the object in the gripper. Examples of the images are shown in Fig. 2.

In practice, the correspondence between the position of the end-effector and the pixel coordinate is not exact. Moreover, the position is three-dimensional whereas the pixel coordinate is two-dimensional, which means there exists a degenerated axis, namely, the optical axis of the camera. Thus, given only a cropped image, it is not possible to determine the correction action. We solve this problem by adding the tentative action a_t^1 to the input state of π^2, so that the correction can depend on the tentative action.

In total, the transformation function is given by

$$s_t^2=f(s_t,a_t^1)=(o_t^2,e_t^2,z_t,a_t^1). \tag{5}$$

Correction function The correction function g performs correction a_t^2 on a_t^1. A straightforward definition is $g(a_t^1,a_t^2)=a_t^1+ca_t^2$, which simply adds the two actions with some coefficient c. However, this simple definition usually violates the range of the action space that $a_t\in[-1,1]$. Considering that an action is squashed into $[-1,1]$ using $\tanh$ in SAC, a natural definition is to sum the two actions with their unsquashed values, namely,

$$g(a_t^1,a_t^2):=\tanh(u_t^1+u_t^2) \tag{6}$$

with $u_t^i = \tanh^{-1}(a_t^i)$ for $i = 1, 2$. In this case, $a_t = g(a_t^1, a_t^2) \in [-1, 1]$ is satisfied.

Another possible definition of g is $g(a_t^1, a_t^2) = a_t^2$. In this case, instead of predicting a correction, the policy of the second stage predicts the final action directly. In particular, as we discussed in the definition of the transformation function f, it is reasonable to include the tentative action a^1 as the input state to π^2. Thus, there is no fundamental difference between predicting the correction or the final action. However, we show that predicting correction is better in practice in the experiment.

B. Train of the two-stage policy

Given the process to execute the policies π^1 and π^2, we now consider the training process. We first present the training based on CQL, and the training based on BC is straightforward. For CQL, we denote the corresponding Q-functions by Q^1 and Q^2.

The first tentative policy π^1, while we name it tentative, is simply a policy that solves the MDP. In particular, we can set the RL objective as $\mathbb{E}_{s_t, a_t \sim \rho_{\pi^1}}[\sum_t \gamma^t r(s_t, a_t)]$, and the training of π^1 and Q^1 is straightforward by applying CQL using the replay data $(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$.

For the second correction policy π^2, the RL objective is given by $\mathbb{E}_{\rho_{\pi^2 \circ \pi^1}}[\sum_t \gamma^t r(s_t, a_t)]$. In principle, it is possible that π^2 depends on π^1 or vice versa. For example, π^1 has some bias in its output action, and π^2 learns the bias and performs better correction. However, it is also possible that π^2 performs correction independent of π^1. In this paper, we consider the case that π^2 is independent of π^1, and replace π^1 by a fixed policy π'. The RL objective then becomes $\mathbb{E}_{\rho_{\pi^2}}[\sum_t \gamma^t r(s_t^2, a_t^2)]$, where we use π', f, and g^{-1} on the state and action s_t, a_t to obtain s_t^2, a_t^2 described as follows.

Let $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ be a replay data sampled from dataset $\mathcal{D}$. Here, in addition to the usual replay data, we assume the next action a_{t+1} is also stored. First, we sample a tentative action a_t^1 according to π'. While the definition of π' can be arbitrary, we assume that $\pi'(\cdot|s_t)$ is close to the action a_t, because a policy trained in an offline setting is close to the behavior policy that generates the dataset $\mathcal{D}$. In particular, we sample a_t^1 by $a_t^1 = \tanh(\tanh^{-1}(a_t) - \epsilon)$ with ϵ sampled from a normal distribution $N(0, \sigma^2)$. Note that this is basically in the form of $a_t^1 = a_t + \epsilon$, but we use $\tanh$ to guarantee that a_t^1 is in the action space. This replacement can also be regarded as data augmentation, because π' outputs arbitrary actions including actions from π^1. Given a tentative action a_t^1, the state s_t becomes $s_t^2 = f(s_t, a_t^1)$, and the action becomes an action a_t^2 satisfying $a_t = g(a_t^1, a_t^2)$. In particular, considering that the correction function g is defined by Eq. (6), we obtain $a_t^2 = \tanh(\epsilon)$. The next state s_{t+1}^2 is obtained in the same way from s_{t+1} and a_{t+1}. In total, we obtain the replay data $(s_t^2, a_t^2, r_t, s_{t+1}^2)$ for training π^2 and Q^2.

The training of π^1 and π^2 with BC is straightforward. We update π^1 and π^2 with the BC loss instead of the CQL loss using the same replay data in both cases.

Algorithm 1 Training π^2, Q^2

Initialize Q network Q_θ^2, policy π_ϕ^2, demonstration data $\mathcal{D}$
for each iteration **do**
  Sample $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ from $\mathcal{D}$.
  Sample $\epsilon \sim N(0, \sigma^2)$.
  Calculate the actions $a_t^1 = \tanh(\tanh^{-1}(a_t) - \epsilon)$ and $a_t^2 = \tanh(\epsilon)$.
  Calculate the states $s_t^2 = f(s_t, a_t^1)$ and $s_{t+1}^2 = f(s_{t+1}, a_{t+1}^1)$.
  Update Q_θ^2, π_ϕ^2 according to BC or CQL with the replay data $(s_t^2, a_t^2, r_t, s_{t+1}^2)$.
end for

V. EXPERIMENTS

Environment We conduct experiments using RLBench [12], which offers a range of vision-based manipulation tasks. We select the same 8 tasks as in [9], also depicted in Fig. 4, that can be accomplished using only the front camera. The state comprises the RGB image of size 128×128 from the front camera image and the current pose of the end-effector $z_t \in \mathbb{R}^7$. The action is the next pose of the end-effector, which includes the position $x_t \in \mathbb{R}^3$, the orientation $r_t \in \mathbb{R}^3$, and the gripper state g_t where open is indicated by $g_t \geq 0$ and close by $g_t < 0$. Given the action, path planning is performed using the 'EndEffectorPoseViaPlanning' action mode provided by RLBench. The reward is 1 on task completion and 0 otherwise.

For the demonstrations, we first generate 100 trajectories using the function provided by RLBench, which is based on pre-defined waypoints for each task. The trajectories are then transformed into replay data using the keyframe discovery and demo augmentation proposed in [9] and stored in the offline dataset. In addition, since we train in an offline setting, the quality of the demonstration data is crucial. We found that the demonstrations generated by the keyframe discovery proposed in [9] contain short transitions, specifically, transitions where the difference in the position part of the action $|x_t - x_{t+1}|$ is small. Such transitions cause the policy to tend to output actions that move slightly or almost stop the end-effector. Thus, we remove such transitions by adding a rule to the rules of keyframe discovery in [9], namely, if the position parts of the actions of two keypoints are close, remove the first keypoint. This modification significantly increases the success probability, especially for BC. Our result with vanilla BC outperforms the one presented in [9], even though we do not use point cloud data in our experiment.

Implementation We train π^1 and π^2 independently. For the training of π^1 and Q^1 with CQL, we use vanilla CQL except that we use the Monte Carlo (MC) return instead of the TD error, namely, replacing the target Q-function with the sum of rewards after time step t in Eq. (3). We also use the evaluation of π^1 as a baseline, which we call CQL(1). We use a 4-layer convolutional neural network (CNN) to extract features from the image o_t, and a 2-layer multi-layer perceptron (MLP) for z_t. The features are concatenated and

used to predict Q values and the parameters of the action distribution with a 2-layer MLP.

For the training of π^2 and Q^2 with CQL, we transform the replay data as Algo. 1 and train using CQL with the MC return. The crop size for o_t^2, the image cropped with the tentative action, is 32×32, and the crop size for e_t^2, the image cropped with the current end-effector pose, is 16×16. We sample $\epsilon \sim N(0, \sigma^2 = 0.1^2)$ for the position and orientation part x_t, r_t, and $\epsilon \sim N(0, \sigma^2 = 1)$ for the gripper state part g_t. We choose a larger σ because the gripper state requires a larger correction, as their values in the dataset are discrete as $g_t = \pm 1$. Note that $\tanh^{-1}(\pm 1)$ is not finite, and we always clip the values in $[-0.99, 0.99]$ before applying $\tanh^{-1}$. We use a 3-layer CNN to extract features from the images o_t^2, e_t^2 individually, and a 2-layer MLP for z_t, a_t^1 individually. The features are concatenated and used to predict Q values and the parameters of the action distribution with a 2-layer MLP.

For the training of π^1 and π^2 with BC, the parameters are the same as CQL except the following. We change the crop size for o_t^2 to 64×64. We use a deterministic policy for BC, and we predict the action itself instead of the parameters for its distribution.

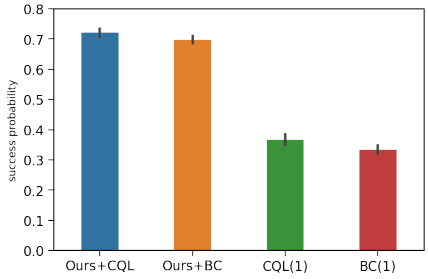

Fig. 3. The average success probability for the 8 RLBench tasks. We evaluate each task with 5 runs of different seeds and 100 episode per run, and the error bars are the stratified bootstrap confidence interval. The result is evaluated at $200,000$ steps for CQL and $300,000$ steps for BC.

Results The main result is presented in Fig. 3 and Fig. 4. The CQL(1) and BC(1) denote the vanilla CQL and BC that have no correction stage. For the two-stage policy, we use the policy trained in CQL(1) and BC(1) as the first stage respectively, and only train the second stage. The result shows that by adding the correction agent, the success probability of these tasks is improved for both CQL and BC for more than 30% in average. The learning curves in Fig. 4 show that while the performance of BC increases with larger training steps, the performance of CQL decreases in some tasks. This is a common problem in CQL, and early stopping is known to be important [23]. Thus, the evaluations in this section are performed at $200,000$ training steps for CQL and $300,000$ training steps for BC.

Next, we study the effect of the crop size. We evaluate with the crop size of o_t^2 for 16×16, 32×32, and 64×64. The results are presented in Fig. 5. For CQL, the success probability for the crop size 32 is slightly better than 64, whereas the crop size 16 is much lower. For BC, the success probability increases as the crop size increases. One common trend is that small crop size performs worse. There are mainly two possible reasons. The first reason is that if the crop size is too small, the range of correctable actions also becomes small, because a small error in the tentative action makes the task-relevant pixels outside of the cropped image. The second reason is that in many cases, while the most task-relevant pixels are near the end-effector, they are not exactly at the position of the end-effector. For example, if the task is to place an object on a table, while the most task-relevant pixels are on the table, the end-effector position is always slightly higher than the table. On the other hand, a large crop size makes the training and inference much slower. Moreover, task-non-relevant information such as the background increases. In general, the size of the end-effector is one metric for the crop size because it corresponds to the amount of task-relevant pixels in many tasks. However, the crop size also depends on the task itself, for example, if the object being manipulated is large, the amount of task-relevant pixels may increase. Moreover, the crop size depends on the accuracy of π^1 in general, as the crop size determines the range of correctable actions. It is not obvious how to determine a good crop size, and a method for automatically tuning the crop size remains as a future work.

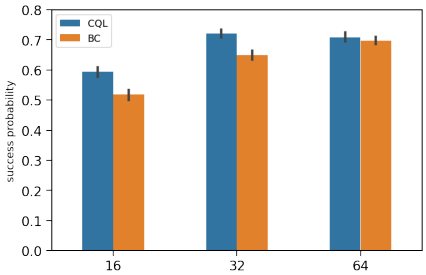

Fig. 5. Comparison of the crop size, measured by the average success probability for the 8 RLBench tasks.

Fig. 6 shows the comparison between predicting the correction and predicting the final action in the second stage. The results show that it is better to predict the correction to the tentative action. The two methods only differ in the definition of the action of the second stage, and the only modification in training is that we set $a_t^2 = a_t$ instead of $a_t^2 = \tanh(\epsilon)$ in Algo. 1. However, this modification changes the distribution of a_t^2. When we use the policy to predict the final action directly, the action values are in a fixed set which has only a few hundred values due to the small size of the offline dataset. On the other hand, when we use the policy to predict the correction action, the distribution of the action values is a tanh-squashed version of the normal distribution, which can be considered to be a more balanced distribution. This result indicates that it is better to predict a more balanced distribution, similar to the cases in classifications and regressions [24], [25].

We also evaluate the difference caused by the usage of the TD error or the MC return. While the performances for both cases are similar for the first stage, we find that the success probability decreases slightly ($\sim 3\%$) for the correction

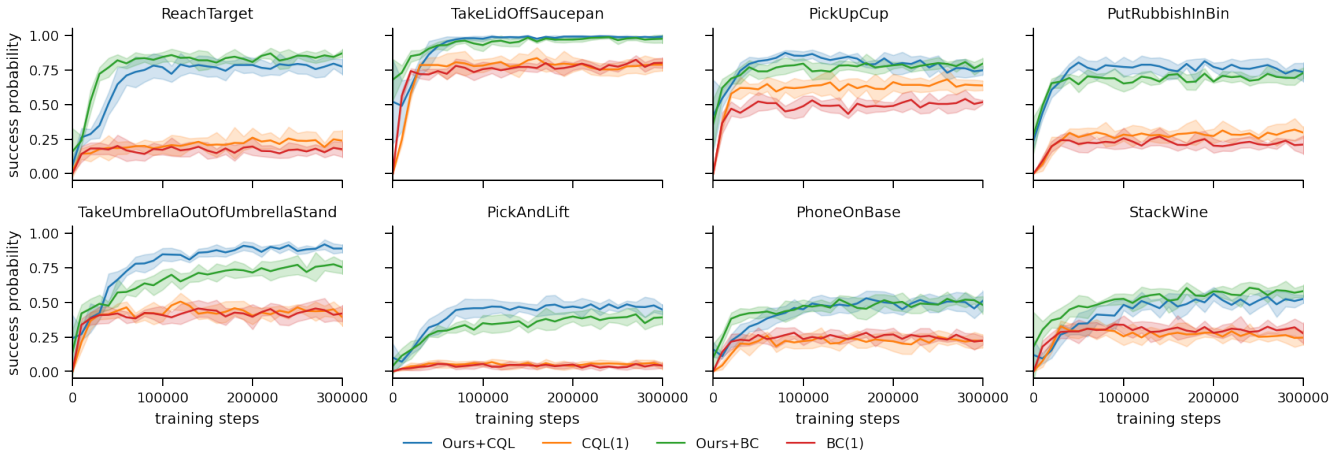

Fig. 4. Learning curves for the 8 RLBench tasks. The solid lines and the shaded regions represent the mean and the standard deviation across 5 runs.

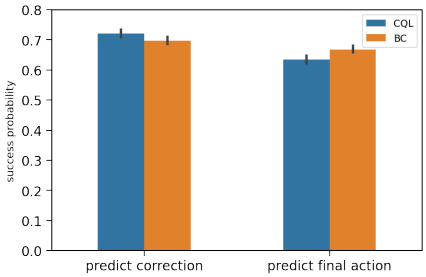

Fig. 6. Comparison of the action predicted in the second stage, measured by the average success probability for the 8 RLBench tasks.

stage if we use the TD error. The primary reason for this performance drop is the instability in training, namely, there are some temporal drops in success probability during the training if we use the TD error. While this performance drop is temporary, it is difficult to determine whether the performance has dropped if we perform offline learning, as evaluation during training is usually not possible. A possible solution for stabilizing the performance of using the TD error is to sample multiple target Q values and use their average, with an increase of the computation cost.

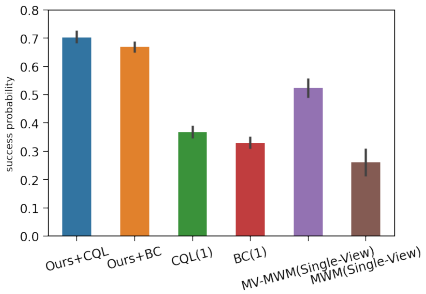

Fig. 7. Comparison to online methods. The results show the average success probability for the 5 RLBench tasks used in [26]. The results for MV-MWM and MWM are sourced from [26].

While we use vanilla CQL and BC as baselines for the offline methods, we also compare the offline methods with the online methods in Fig. 7. Two online methods, the Multi-View Masked World Model (MV-MWM) [26] and the Masked World Model (MWM) [27], are chosen as they are also evaluated with RLBench and use RGB images as the input, although there are many differences in the other configurations of the environment. We see that the proposed method achieves a higher success probability with less data because our method is offline. Note that we only compare the result with a single camera configuration in [26], which uses two cameras, the front camera and the wrist camera, for learning the visual representation and a single camera, the front camera, for RL. The result for MV-MWM with two cameras for RL is superior than ours. It requires additional effort to implement our method for the wrist camera, as the camera moves with the end-effector and the correspondence between the position of the end-effector and the pixel coordinate is changing. One reason that our offline methods outperform the online ones is that we explicitly utilize the property of the manipulation tasks to crop the image. A natural limitation to our proposed method is that it is basically only applicable to manipulation tasks.

VI. CONCLUSIONS

We have presented a two-stage agent with a tentative decision stage and a correction stage, both of which can be trained individually. We conducted experiments with 8 RLBench tasks to show that the correction stage improves the task success probability in the offline setting, and the results were superior to those of the online methods.

An important property utilized in this paper is the relationship between the action and the crop position, which is specific to manipulation tasks. While it is straightforward to determine the crop position, it is not trivial to determine the crop size, which may depend on the tasks. In online settings, we can learn the crop size directly as an action in a similar manner to [5], [6], but this is not applicable in offline settings. A method for determining the crop size remains as a future work.

REFERENCES

[1] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, *et al.*, "Rt-1: Robotics transformer for real-world control at scale," *arXiv preprint arXiv:2212.06817*, 2022.

[2] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.

[3] D. Kalashnikov, A. Irpan, P. P. Sampedro, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," in *Conference on Robot Learning*, 2018.

[4] D. Kalashnikov, J. Varley, Y. Chebotar, B. Swanson, R. Jonschkowski, C. Finn, S. Levine, and K. Hausman, "Mt-opt: Continuous multi-task robotic reinforcement learning at scale," *arXiv preprint arXiv:2104.08212*, 2021.

[5] B. Wu, I. Akinola, and P. K. Allen, "Pixel-attentive policy gradient for multi-fingered grasping in cluttered scenes," in *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2019, pp. 1789–1796.

[6] B. Wu, I. Akinola, A. Gupta, F. Xu, J. Varley, D. Watkins-Valls, and P. K. Allen, "Generative attention learning: A "general" framework for high-performance multi-fingered grasping in clutter," *Autonomous Robots*, vol. 44, no. 6, pp. 971–990, 2020.

[7] M. Gualtieri and R. Platt, "Learning 6-dof grasping and pick-place using attention focus," in *Conference on Robot Learning*. PMLR, 2018, pp. 477–486.

[8] M. Gualtieri and R. Platt, "Learning manipulation skills via hierarchical spatial attention," *IEEE Transactions on Robotics*, vol. 36, no. 4, pp. 1067–1078, 2020.

[9] S. James and A. J. Davison, "Q-attention: Enabling efficient learning for vision-based robotic manipulation," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1612–1619, 2022.

[10] S. James, K. Wada, T. Laidlow, and A. J. Davison, "Coarse-to-fine q-attention: Efficient learning for visual robotic manipulation via discretisation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 13 739–13 748.

[11] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *arXiv preprint arXiv:1812.06298*, 2018.

[12] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3019–3026, 2020.

[13] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 1179–1191.

[14] D. Yarats, I. Kostrikov, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," in *International Conference on Learning Representations*, 2021.

[15] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv preprint arXiv:2005.01643*, 2020.

[16] R. F. Prudencio, M. R. O. A. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[17] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, "Combo: Conservative offline model-based policy optimization," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 28 954–28 967.

[18] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 20 132–20 145.

[19] A. Nair, A. Gupta, M. Dalal, and S. Levine, "Awac: Accelerating online reinforcement learning with offline datasets," *arXiv preprint arXiv:2006.09359*, 2020.

[20] Z. Wang, A. Novikov, K. Zolna, J. S. Merel, J. T. Springenberg, S. E. Reed, B. Shahriari, N. Siegel, C. Gulcehre, N. Heess, *et al.*, "Critic regularized regression," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 7768–7778.

[21] A. Kumar, J. Hong, A. Singh, and S. Levine, "Should i run offline reinforcement learning or behavioral cloning?" in *International Conference on Learning Representations*, 2022.

[22] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1861–1870.

[23] A. Kumar, A. Singh, S. Tian, C. Finn, and S. Levine, "A workflow for offline model-free robotic reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 417–428.

[24] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[25] Y. Yang, K. Zha, Y. Chen, H. Wang, and D. Katabi, "Delving into deep imbalanced regression," in *International Conference on Machine Learning*. PMLR, 2021, pp. 11 842–11 851.

[26] Y. Seo, J. Kim, S. James, K. Lee, J. Shin, and P. Abbeel, "Multi-view masked world models for visual robotic manipulation," in *International Conference on Machine Learning*, 2023.

[27] Y. Seo, D. Hafner, H. Liu, F. Liu, S. James, K. Lee, and P. Abbeel, "Masked world models for visual control," in *Conference on Robot Learning*. PMLR, 2023, pp. 1332–1344.