Reinforcement Learning without Ground-Truth State

Xingyu Lin, Harjatin Singh Baweja and David Held
Robotics Institute
Carnegie Mellon University
{xlin3, harjatis, dheld}@andrew.cmu.edu

Abstract—To perform robot manipulation tasks, a low dimension state of the environment typically needs to be estimated. However, designing a state estimator can sometimes be difficult, especially in environments with deformable objects. An alternative is to learn an end-to-end policy that maps directly from high dimensional sensor inputs to actions. However, if this policy is trained with reinforcement learning, then without a state estimator, it is hard to specify a reward function based on continuous and high dimensional observations. To meet this challenge, we propose a simple indicator reward function for goal-conditioned reinforcement learning: we only give a positive reward when the robot’s observation exactly matches a target goal observation. We show that by utilizing the goal relabeling technique, we can learn with the indicator reward function even in continuous state spaces, in which we do not expect any two observations to ever be identical. We propose two methods to further speed up convergence with indicator rewards: reward balancing and reward filtering. We show comparable performance between our method and an oracle which uses the ground-truth state for computing rewards, even though our method only operates on raw observations and does not have access to the ground-truth state. We demonstrate our method in complex tasks in continuous state spaces such as rope manipulation from RGB-D images, without knowledge of the ground truth state.

I. INTRODUCTION

To perform robot manipulation tasks, a low dimensional state of the environment typically needs to be estimated. In reinforcement learning, this state is also used to compute the reward function. However, designing a state estimator can be difficult, especially in environments with deformable objects, as shown in Figure 1. An alternative is to learn an end-to-end policy that maps directly from high dimensional sensor input to actions. However, without a state estimator, it is hard to specify a reward function based on high dimensional observations.

Past efforts to use reinforcement learning for robotics have avoided this issue in a number of ways. One common approach is to use extra sensors to determine the state of the environment during training, even if such sensors are not available at test time. Examples of this include using another robot arm to hold all relevant objects [23], placing an IMU sensor [16, 46] or motion capture markers on such objects [21], or ensuring that all relevant objects are placed on scales [34].

However, such instrumentation is not always easy to set up for each task. This is especially true for deformable object manipulation, such as rope or cloth manipulation, in which every part of the object must be instrumented in order to measure the full state of the entire object. Attaching such sensors to food or granular material would present additional difficulties.

Another common approach is to train the policy entirely in simulation in which ground truth state can be obtained from the simulator. However, transferring trained policies from simulation to the real world can be challenging [40, 42]. Furthermore, for both the “training in simulation” approach as well as the “training with extra sensors” approach, once the policy is trained, the parameters must be fixed. Because ground truth state is required by these methods for training, they do not enable extra adaptation to be performed once the robot is deployed. This approach is fairly limiting and makes strong assumptions that the training environment matches the testing environment. In the real world, the test environment does not always match the laboratory setting or simulation in which the robot was trained; a more robust system should allow additional test-time adaptation, which is not possible with these previous methods, due to the requirement of needing access to the ground truth state for the reward function.

We present an alternative approach for goal-conditioned reinforcement learning for specifying rewards using raw (e.g. high-dimensional and continuous) observations without requiring explicit state estimation or access to the ground truth state of the environment. We achieve this using a simple indicator reward function, which only gives a positive reward when the robot’s observation exactly matches a target goal observation. Naturally, in continuous state spaces, we do not expect any two observed states to be identical. Surprisingly, we show that we can learn with such an indicator reward, even in continuous...
state spaces, if we use goal relabeling [20, 2]. We use an off-policy reinforcement learning algorithm and replace goals with past observations; using goal relabeling, each observation will also be used as a relabeled goal. We show that we can further reduce the training time using two techniques that we introduce: reward balancing and reward filtering.

We show theoretically that the indicator reward results in a policy with bounded suboptimality compared to the ground-truth reward. We also empirically show comparable performance between our method and an oracle which uses the ground-truth state for computing rewards, even though our method only operates on raw observations and does not have access to the ground-truth state. We demonstrate that an indicator reward can be used to teach a robot complex tasks in continuous state spaces such as rope manipulation from RGB-D images, without knowledge of the ground truth state during training.

II. RELATED WORK

A. Obtaining ground truth state for training

Adding sensors: To compute rewards for reinforcement learning, most algorithms assume access to the ground truth state. However, in most real world settings, the ground truth state is not directly observed but must be estimated using a noisy state estimator, which can lead to noisy rewards and poor learning. Attempts have been made to avoid this issue by adding extra sensors during training to accurately record the state. For example, in past work, one robot arm (covered with a cloth at training time) is used to rigidly hold and move an object, while another robot arm learns to manipulate the object [23]. In such a case, the object position can be inferred directly from the position of the robot gripper that is holding it. In other work on teaching a robot to open a door, an IMU sensor is placed on the door handle to determine the rotation angle of the handle and whether or not the door has been opened [16, 46]. One can also ensure that all relevant objects for a task are placed on scales [34] or affixed with motion capture markers to obtain a precise estimate of their position [21]. However, such instrumentation is challenging for deformable objects, granular material, food, or other settings. Further, such instrumentation is costly and time-consuming to setup; hence most of these previous approaches assume that such instrumentation is only available at training time and these methods do not allow further fine-tuning of the policy after deployment.

Training in simulation: Another work-around to the issue of noisy state estimation is to train the policy entirely in simulation, in which the ground truth state can be obtained from the simulator [11, 3, 47, 32, 31]. Many approaches have been explored to try to transfer such a policy from simulation to the real world, such as domain randomization [42] or building or learning a more accurate simulator [40, 6]. However, obtaining an accurate simulator is often very challenging; especially if the simulator differs from the real-world in unknown ways, these methods will not transfer well to the real world. Further, building the simulator itself can be fairly complex. Because these methods require the ground truth state to obtain the reward function, they require training in a simulator and do not allow further fine-tuning after deployment in the real world; our method, in contrast, does not require the ground truth state for the reward function.

B. Robot learning without ground truth state

Learning a reward function without supervision: One line of work for learning a reward function is to first learn a latent representation and then use the L2 distance or cosine similarity in the embedding space as a reward. Different approaches for representation learning have been explored, such as maximizing the mutual information between the achieved goal and the intended goal [45], reconstruction of the observation with VAE [26], or learning to match keypoints with spatial autoencoders [14]. Our approach is much simpler in that the reward function does not have any parameters that need to be learned.

Changing the optimization: Another approach is to forego maximizing a sum of rewards as is typically done in reinforcement learning and instead optimize for another objective. For example, one method is to choose one-step greedy actions based on a learned one-step inverse dynamics model; after training, the policy is then applied directly to a multi-step goal [1]. An alternative method is to learn a predictive forward dynamics model directly in a high-dimensional state space and use visual model-predictive control [13, 12, 8, 9, 10]. Although these methods have shown some promise, predicting future high-dimensional observations (such as images or depth measurements) is challenging. Another approach is to obtain expert demonstrations and define an objective as trying to imitate the expert [39, 38, 29, 15]. Our approach, however, applies even when demonstrations are not available.

Incentivizing exploration: An alternative direction is to reward an agent for visiting unexplored parts of its environment [35, 36, 4, 17, 41, 5, 28]. However, if such an approach is not combined with some kind of goal-oriented learning, then the agent will not actually learn how to later perform intentional actions needed to achieve a task. Thus, the question of incentivizing exploration is somewhat orthogonal to the ideas explored in this paper.

C. Manipulating deformable objects

Deformable object manipulation presents many challenges for both perception and control. One approach to the perception problem is to perform non-rigid registration to a deformable model of the object being manipulated [18, 22, 37, 44, 19, 25, 7, 30]. However, such an approach is often slow, leading to slow policy learning, and can produce errors, leading to poor policy performance. Further, such an approach often requires a 3D deformable model of the object being manipulated, which may be difficult to obtain. Our approach applies directly to high-dimensional observations of the deformable object and does not require a prior model of the object being manipulated.
III. PROBLEM FORMULATION

In reinforcement learning, an agent interacts with the environment over discrete time steps. In each time step, the agent observes the current state $s_t$ and takes an action $a_t$. In the next time step, the agent transitions to a new state $s_{t+1}$ based on the transition dynamics $p(s_{t+1}|s_t, a_t)$ and receives a reward $r_{t+1} = r(s_{t+1}, a_t, s_{t+1})$. The objective for the agent is to learn a policy $\pi(a_t|s_t)$ that maximizes the expected future return $R = \mathbb{E} \left[ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \right]$, where $\gamma$ is a discount factor in the range of $[0, 1]$.

A. Goal-reaching Reinforcement Learning

In order for the agent to learn diverse and general skills, we define a goal reaching MDP [33, 2] as follows: In the beginning of each episode, a goal state $s_g$ is sampled from a goal distribution $\mathcal{G}$. We learn a goal conditioned policy $\pi(a_t|s_t, s_g)$ that tries to reach any goal state from the goal distribution. We use a goal conditioned reward function $r_t = r(s_{t+1}, s_g)$ and optimize for $\mathbb{E}_{s_t \sim \mathcal{G}} \left[ \sum_{t=0}^{\infty} \gamma^t r_t \right]$. The transition dynamics $p(s_{t+1}|s_t, a_t)$ of the environment remain independent to the goal.

In many real-world scenarios, it is often difficult to construct a well-shaped reward function. Past work has shown that sparse rewards, combined with an appropriate learning algorithm, can achieve better performance than poorly-shaped dense rewards in goal-reaching environments [2]. We thus define a sparse reward function that only makes the binary decision of whether the goal is reached or not. Specifically, let $S_+(s_g)$ be a subset of the state space such that any state in this set is determined to be sufficiently close to $s_g$ (in some unknown metric); in other words, if the environmental state is within $S_+(s_g)$, then the task of reaching $s_g$ can be considered to be achieved (note that $S_+(s_g)$ can be defined to be an arbitrarily small set, depending on the task specifications). Naturally, we can assume that $s_g \in S_+(s_g)$. A binary reward function can then be defined as

$$r(s_{t+1}, s_g) = \begin{cases} R_+ & s_{t+1} \in S_+(s_g) \\ R_- & s_{t+1} \notin S_+(s_g), \end{cases}$$

where $R_+$ and $R_-$ are constants representing the rewards received for achieving the goal and failing to achieve the goal, respectively.

IV. APPROACH

A. Proxy Reward Functions

In many cases, the ground truth state $s_t$ is unknown and we cannot directly use the true reward function defined in Equation 1. Instead, the agent observes high dimensional observations $o_t$ from sensors, from which we must instead define a proxy reward function $\hat{r}(o_{t+1}, o_g)$. The question now becomes how to choose $\hat{r}$ to be optimal for reinforcement learning, i.e. which choice of $\hat{r}$ will lead to the fastest learning and most accurate policy?

The most common approach in robotics is to perform state estimation. Let us define an (unknown) function $f$ that maps an observation $o_t$ to its corresponding ground truth state $s_t = f(o_t)$. However, since we do not observe the ground truth state, one approach is to estimate it: $\hat{s}_t = f(o_t)$. This estimated state can then be used as if it were the true state in equation 1:

$$r(\hat{s}_{t+1}, s_g) = \begin{cases} R_+ & \hat{s}_{t+1} \in S_+(s_g) \\ R_- & \hat{s}_{t+1} \notin S_+(s_g), \end{cases}$$

However, this approach has two potential issues: first, the state estimator might be noisy, leading to a noisy reward signal. Second, the state estimator itself might be hard to obtain; this is especially the case for deformable object manipulation.

We therefore investigate whether an alternative reward function that does not depend on state estimation can be used. Specifically, let us consider a general reward function of the form

$$\hat{r}(o_{t+1}, o_g) = \begin{cases} R_+ & o_{t+1} \in \hat{O}_+(o_g) \\ R_- & o_{t+1} \notin \hat{O}_+(o_g), \end{cases}$$

where $o_g$ is a representation of the goal in observation space and $\hat{O}_+(o_g)$ is a subset of the observation space for which we will give positive rewards. For example, if we define $\hat{O}_+(o_g)$ using noisy state estimation, i.e. $\hat{O}_+(o_g) = \{ o_t : \hat{f}(o_t) \in S_+(s_g) \}$, then we would recover the reward function of Equation 2; however, if we do not have access to a state estimator (e.g. for deformable object manipulation), then we must find another choice of $\hat{O}_+(o_g)$. Next we will investigate tradeoffs between different choices of $\hat{O}_+(o_g)$ and how they will affect policy training time when trained with rewards of $\hat{r}(o_{t+1}, o_g)$.

B. Reward Misclassifications

We will now investigate how to design a good proxy reward function $\hat{r}(o_{t+1}, o_g)$, based on raw sensor observations, that we can use to train the policy; we desire for the policy trained with $\hat{r}(o_{t+1}, o_g)$ to optimize the original reward $r(s_{t+1}, s_g)$ based on the ground-truth state (which we do not have access to). Our first insight into choosing a good proxy reward function $\hat{r}(o_{t+1}, o_g)$ is that we should think about reward functions in terms of false positives and false negatives. Let us define a false positive reward to occur when the agent receives a positive reward based on our proxy reward function $\hat{r}(o_{t+1}, o_g)$ when it would have received a negative reward based on the original reward function $r(s_{t+1}, s_g)$. In other words, a false positive reward occurs when $o_{t+1} \in \hat{O}_+(o_g)$ while $f(o_{t+1}) \notin S_+(s_g)$. Similarly, a false negative reward occur when the agent receives a negative reward based on our proxy reward function $\hat{r}(o_{t+1}, o_g)$ when it would have received a positive reward based on the original reward function $r(s_{t+1}, s_g)$.

Intuitively, both false positive rewards and false negative rewards can negatively impact learning. However, for any estimated reward function $\hat{r}(o_{t+1}, o_g)$, we will have either false positives or false negatives (or both) unless we have access to a perfect state estimator. Assuming that we do not

\footnote{$S_+$ is a function that maps from the state space to a subset of the space.}
have a perfect state estimator, we must ask: which will more negatively affect learning: false positives or false negatives?

The two types of mistakes are not symmetric. As we will see, a false positive reward can significantly hurt policy learning, while a false negative reward is much more tolerable. Under a false positive reward, the agent receives a positive reward (under the proxy reward function $\hat{r}(o, o_g)$) for reaching some observation $o$, even though the agent should receive a negative reward based on the corresponding ground-truth state $f(o)$ under the original reward function $r(s_t, s_g)$. This false positive reward will encourage the agent to continue to try to reach the state $f(o)$, even though reaching this state does not achieve the original task since $f(o) \notin S_+(s_g)$.

On the other hand, false negative rewards are much more tolerable. Under a false negative, the agent observes some observation $o$ such that $f(o) \in S_+(s_g)$ but the agent receives a negative reward. However, if the agent still receives a positive reward for some other observation $o'$ such that $f(o') \in S_+(s_g)$, then the agent can still learn to reach the goal states $S_+(s_g)$, though learning might be slower and the learned policy may be suboptimal.

We provide a simple example to verify this intuition. Consider a robot arm reaching task, with the observation space $O \subset \mathbb{R}^3$ being the 3D position of the end-effector (EE). The action space $A \subset \mathbb{R}^3$ controls the position of EE. The true reward is defined by $O_+(o_g) = \{ o \mid \| o - o_g \| < \epsilon \}$. We define two types of noisy reward functions used for training. The reward function $\hat{r}_{FP}$ gives the same rewards as the true reward function, except that with a probability of $p_{FP}$ (False Positive Rate), a negative reward will be flipped to a positive reward. The reward function $\hat{r}_{FN}$ can be similarly defined, where a positive reward will be flipped to a negative reward with a probability of $p_{FN}$ (False Negative Rate). For this experiment, we use a standard reinforcement learning algorithm DDPG [24] combined with goal relabeling [2]. The learning performance of this same algorithm with different noisy rewards can be shown in Figure 2. We can see that the agent is able to learn the task even with a very large false negative rate. But when the false positive rate increases, the performance sharply decreases.

Following this idea, we propose using a proxy reward function that does not have any false positive rewards. To do so, we will use an extreme reward function of $\hat{O}_+(o_g) = \{ o_g \}$. In other words, we will use an indicator reward function:

$$
\hat{r}_{ind}(o_{t+1}, o_g) = \begin{cases} 
R_+ & o_{t+1} = o_g \\
R_- & o_{t+1} \neq o_g,
\end{cases}
$$

It should be clear that this reward function will have no false positives, since the reward is positive only if $o_{t+1} = o_g$, which implies that $f(o_{t+1}) = f(o_g)$, or equivalently, $s_{t+1} = s_g$. As $s_g \in S_+(s_g)$ by definition, all positive rewards are true positives, and there will be no false positive rewards. However, this reward function is extremely sparse and has many false negatives. In fact, without goal relabeling, in continuous state spaces, we would expect all rewards to be negative under this indicator reward function, as no two observations in continuous spaces will ever be identical. Next, we will describe how to learn with this reward function with goal relabeling.

C. Goal Relabeling for Off-policy learning

Fortunately, for off-policy multi-goal learning, we can adopt the goal relabeling technique introduced in [20, 2] to learn the goal-conditioned Q function. Suppose that some transitions $(o_t, a_t, o_{t+1})$ are observed when the agent takes an action $a_t$, which will receive a positive reward under our indicator reward function. With probability $1 - p_1$, we use the

<Fig. 2: Compared to false negatives, false positives can be much worse for learning a good policy. The y-axis shows the success rate of the learned policy after convergence, defined as the frequency of reaching the goal configuration.>

continuous spaces. Specifically, for some transitions, we will choose to replace $o_g$ with another observation $o_{t+1}$ and using our indicator reward function, we will have that $\hat{r}_{ind}(o_{t+1}, o_g) = \hat{r}_{ind}(o_{t+1}, o_{t+1}) = R_+$. Thus, using goal relabeling, we can get positive rewards, even when using an indicator reward function in continuous state spaces.

D. Reward Balancing and Filtering

Our full algorithm for learning with an indicator reward function is shown in Algorithm 1. After sampling a batch of data, we train the Q-function with goal relabeling. We use three different strategies for goal relabeling: with probability $p_1 p_2$, we relabel $o_g$ with $o_{t+1}$, which will receive a positive reward under our indicator reward function. With probability $p_1 (1 - p_2)$, we relabel the goal $o_g$ with $o_t$, with an observation from some future time $t'$ step within the episode. The indicator reward function will most likely give a negative reward in this case, which is possibly a false negative (which we assume to be tolerable). Finally, with probability $1 - p_1$, we use the
original goal (with no relabeling), which will again most likely give a negative reward under the indicator reward function; as before, this might be a false negative.

We refer to “reward balancing” as setting \( p_1 = 0.9 \) and \( p_2 = 0.5 \), leading us to receive positive rewards approximately 0.45 of the time and negative rewards approximately 0.55 of the time. Thus the ratio of positive and negative rewards that we use to train the Q-function are approximately balanced, even with indicator rewards. From another perspective, \( p_2 \) can be seen as a weighting factor between providing positive rewards and propagating rewards to other timesteps in the episode. Additionally, training with a small fraction of the original goals (i.e. \( 1 - p_1 \)) can be seen as a regularization which ensures that the distribution of the relabeled goals moves towards the original goal distribution.

**Reward filtering** While false negative rewards do not hurt learning as much as false positives, we still wish to avoid them if possible to improve the convergence time of the learned policy. We achieve this using “reward filtering,” in which we filter out transitions that we suspect of having a high chance of being false negatives. As shown in line 24, we discard a sampled transition if its Q value is above a threshold \( q_0 \). If the assigned reward is negative based on the proxy reward function (Equation 1) will minimize \( Q^* (o_t, a_t, \hat{O}_+ (o_g)) \), we find that \( Q^* (o_t, a_t, \hat{O}_+ (o_g)) = R_+ / (1 - \gamma) \), where \( Q^* \) is the optimal Q-function, assuming the optimal policy will continue to receive (discounted) positive rewards in the future. Similarly, if \( o_{t+1} \neq o_g \), then \( \hat{r}_{ind} (o_{t+1}, o_g) = R_- \).

Since we know that the policy starting from \( o_t \) will thus receive at least one negative reward before receiving positive rewards, then \( Q^* (o_t, a_t, o_g) \leq R_- + \gamma R_+ / (1 - \gamma) \). Thus, we can set a threshold \( q_0 \), where \( R_- + \gamma R_+ / (1 - \gamma) < q_0 < R_+ / (1 - \gamma) \); if we find that \( Q(o_t, a_t, o_g) > q_0 \), then the corresponding reward \( \hat{r}_{ind} (o_{t+1}, o_g) \) is likely to be a false negative (assuming that the Q-function has been trained well); we thus filter out such rewards, to reduce the number of false negatives that we use for training.

**V. Analysis**

In this section, we analyze the performance of learning with indicator rewards. As a start, we first interpret the goal conditioned Q function as a measure of the time it takes for the agent to reach one observation from another.

**A. Minimum Reaching Time Interpretation**

Let us define \( d = D_+ (o_t, a_t, \hat{O}_+ (o_g)) \) as the number of time steps it takes for the policy \( \pi \) to go from the current observation \( o_t \), starting with action \( a_t \), to reach the set \( \hat{O}_+ (o_g) \) of goal observations. For simplicity, we assume that, once the agent receives a positive reward, it will take actions to continue to receive positive rewards. The Q function can be written as

\[
Q_\pi (o_t, a_t, o_g) = R_- + \gamma R_- + ... + \gamma^{d-1} R_- + \gamma^d R_+ + \gamma^{d+1} R_+ + ... = \frac{\gamma^d (R_+ - R_-) + 1}{1 - \gamma} R_- \tag{6}
\]

Now it can be easily seen that, as long as \( R_+ > R_- \), \( Q_\pi \) is strictly monotonically decreasing w.r.t. \( d \). As such, maximizing \( Q_\pi \) over \( \pi \) is equivalent to minimizing the time the agent takes to reach the goal \( O_+ (o_g) \). Note that this is true for varying definitions of \( O_+ (o_g) \); thus the policy trained under the true reward function (Equation 1) will minimize \( D_+ (o_t, a_t, O_+ (o_g)) \) whereas the policy trained under the indicator reward function will minimize \( D_\pi (o_t, a_t, o_g) \) (slightly overloading notation for \( D_\pi \)). Below we will show how this interpretation of the policy’s behavior at convergence can lead to a simple analysis of the suboptimality of the learned policy under the indicator reward.

### Algorithm 1: Learning with Indicator Reward Function

1. \( R \): Replay buffer.
2. \( \pi_\theta \): Policy to be learned.
3. \( \pi_\beta \): Behaviour policy.
4. \( \hat{r}_{ind} : O \times O_g \rightarrow \mathbb{R} \): Indicator reward function
5. for \( i \leftarrow 1 \) to \( N_{epoch} \) do
   for \( j \leftarrow 1 \) to \( N_{cycle} \) do
     Sample an initial observation \( o_0 \) and a goal \( o_g \)
     Collect \( \tau = (o_0, a_0, ..., o_T, a_T) \) following \( \pi_\beta \) and goal \( o_g \)
     Store all transitions \((o_t, a_t, o_g, r_{t+1}, o_{t+1}, t)\) in \( R \)
   end
   for \( k \leftarrow 1 \) to \( N_{train} \) do
     Sample a mini-batch \( B \) from the replay buffer
     for each transition in \( R \) do
       With probability \( p_1 \) // \( p_1 = 0.9 \)
       \( o_t \leftarrow o_{t+1} \)
       \( r_{t+1} \leftarrow \hat{r}_{ind} (o_{t+1}, o_g) \) // \( r_{t+1} = R_+ \)
     else:
       Sample a future time step \( t' \) from \( \{t + 1, ..., T\} \)
       \( o_{t+1} \leftarrow o_t \)
       \( r_{t+1} \leftarrow \hat{r}_{ind} (o_{t+1}, o_g) \) // \( r_{t+1} \approx R_- \)
     end
     If \( r_{t+1} = R_- \) and \( Q(o_t, a_t, o_g) > q_0 \):
       Discard this transition
   end
   Perform one step of optimization using TD(0)
end

---

---
B. Analysis of Sub-optimality

Due to the false negative rewards given by the indicator function \( \hat{r}_{ind} \), the learned policy may not be optimal with respect to the original reward function \( r(s_{t+1}, s_g) \) defined in Equation 1. Here we give the worst case bound for the policy learned with the indicator reward. Following the minimum reaching time interpretation of the previous section, we evaluate the performance of the policy in terms of the time it takes to reach the set of goal observations \( O_+(o_g) \) from the current observation. Given \( o_t, o_g \), denote \( t_1 \) as the minimum number of time steps to reach from \( o_t \) to the set of true goal observations, i.e. \( t_1 = D(o_t, O_+(o_g)) \). Let \( t_2 = D(o_t, o_g) \) be the minimum time to reach from \( o_t \) to \( o_g \) (note that we are somewhat overloading notation for \( D \)). Define the diameter of this goal observation set as \( d = \max\{D(o_1, o_2) | o_1, o_2 \in O_+(o_g)\} \).

From the optimality of \( t_2 \), we know that

\[
t_2 \leq D(o_t, o) + D(o, o_g), \forall o \in O_+(o_g).
\]

Thus,

\[
t_2 \leq \min_{o \in O_+(o_g)} D(o_t, o) + D(o, o_g) \leq t_1 + d.
\]

From the analysis in the previous section, the optimal policy which optimizes the indicator reward will reach \( o_g \) in \( t_2 \) time steps; since \( o_g \in O_+(o_g) \), we know that this policy will reach \( O_+(o_g) \) in some time \( t_3 \leq t_2 \). Also recall that we have defined \( t_1 \) such that the optimal policy under the true reward function of Equation 1 will reach \( O_+(o_g) \) in \( t_1 \) steps. Thus \( t_3/t_1 \leq (t_1 + d)/t_1 \) is an upper bound on the suboptimality of the policy trained under the indicator reward, at convergence.

VI. EXPERIMENTS

Our experiments address the following questions:

1) Is the robot able to learn to reach goals given only the indicator rewards?
2) In the case of visual input, how much are the sample efficiency and the final performance affected without assuming access to the ground truth reward?
3) How much does reward balancing and filtering improve learning efficiency?
4) Can our method scale to real world robotic tasks?

We denote our method, which uses indicator rewards with reward balancing and filtering, as \textbf{Indicator+Balance+Filter}.

We compare our method with the following methods:

- **Oracle**: This method assumes access to the ground truth reward \( r(s_t, s_g) \).
- **Auto Encoder**: For vision-based tasks, we train an autoencoder with an \( L^2 \) reconstruction loss of the image observation, jointly with the RL agent. We then use cosine similarity in the learned embedding space to provide dense rewards, as similarly compared in [45]. Specifically, assuming the learned encoding of an observation \( o \) is \( \phi(o) \) after \( L^2 \) normalization, the reward will be \( r(o, o_g) = \max(0, \phi(o)^T \phi(o_g)) \).

- **Indicator**: This is the same as our method, without reward balancing and filtering.

We use the standard off-policy learning algorithm DDPG [24] with goal relabeling [2]. For the baselines, we use the goal-relabeling and sampling strategy that result in the best performance, for a fair comparison. Specifically, for all these methods, with a probability of 0.9, we re-label the current goal with an achieved goal sampled uniformly from one of the future time steps within the same episode; otherwise, the original goals are used. For all the environments, the ground truth rewards are based on the \( L_2 \) distance in the state space:

\[
r(s_{t+1}, s_g) = \begin{cases} R_+ & \|s_{t+1} - s_g\| \leq \epsilon \\ R_- & \text{o.w.} \end{cases}
\]

More details on the algorithm can be found in Appendix B.

A. Learning with Indicator Rewards

We first evaluate all the methods in a set of simulated environments in MuJoCo [43], where all the methods receive the state representation as input:

- **Reacher**: Teach a two-link arm to reach a randomly located position in 2D space.
- **FetchReach**: Move the end effector of the Fetch robot to a random position in 3D.
- **FetchPush**: Use the Fetch robot to push a block to a targeted location.

The three environments above are relatively easy, standard environments from Gym [27]. Additionally, we test our algorithms in a more complex RopePush environment, where the robot needs to push a 15-link rope to a targeted pose, as shown in Figure 1. To accelerate learning, we fix the orientation of the gripper and parameterize the action as \( (x_1, y_1, x_2, y_2) \in \mathbb{R}^4 \), denoting the starting and ending position of one push from the gripper. We generate each initial rope pose by giving the rope a random push from a fixed location. The goal poses are each generated by giving the rope two more pushes based on the initial push. When we evaluate the policy, we allow the robot three pushes to reach the goal pose from the initial pose. More details on the test environments can be found in Appendix A.

The performance of different learning methods are shown in Figure 3. In all of the environments, using indicator rewards with reward balancing and filtering achieves comparable performance to Oracle. Our method also achieves much better sample efficiency than the ablation which uses the indicator reward without balancing or filtering. In Appendix C, we perform a further ablation analysis to determine the separate effect of reward balancing and reward filtering. Our results show that both reward balancing and filtering are required for optimal performance.

Interestingly, our method often outperforms the Oracle in reducing the distance to the goal (Figure 3, bottom). The reason for this is as follows: in the FetchReach environment, the default distance threshold \( \epsilon \) for receiving an \( R_+ \) reward is set to 0.05 m. Thus, the policy that learns with this reward is not incentivized to bring the agent closer than 0.05 m to
the goal. On the other hand, the policy learned with indicator rewards continues to reduce the distance to the goal. This shows another benefit of using the indicator reward: the user does not need to tune the hyper-parameter $\epsilon$ to achieve optimal performance.

B. Learning with Indicator Rewards from Simulated Images

Next, we test all the algorithms in the visual Reacher, FetchReach, and RopePush environments where both the current and goal observations are provided as RGB-D images. Figure 4 shows that our method is able to learn at nearly the same rate as the Oracle, even though our method only has access to images and does not use the ground-truth state. Our method also learns faster than the autoencoder baseline and the ablation without balancing or filtering; the difference is especially significant in the most complex RopePush environment, in which only our method and the Oracle are able to learn the task.

C. Learning with Indicator Rewards from Real Images

Using RGB-D observations and goals, we train a Sawyer robot for a 3 dimensional reaching task. Figure 5 shows the experimental setup (left) as well as example observation and goal images. The results are shown in Figure 6. As before, our method (Indicator+Balance+Filter) performs similarly to the Oracle both in terms of success rate and final goal distance; the baseline of Indicator without balancing or filtering performs significantly worse.

VII. Conclusion

In this work, we show that we can train a robot to perform complex manipulation tasks directly from high-dimensional images, without requiring access to the ground-truth state in either the policy input or the reward function. We empirically show that our method enables a robot to learn complex skills for manipulating deformable objects, for which state estimation is often challenging. We also demonstrate that our method performs well in the real world.

We provide a theoretical analysis which shows that the optimal policy under the indicator reward has a bounded sub-optimality compared to the optimal policy under the ground-truth reward. We hope that our method will enable robot learning in the real world in cases where it is difficult to add extra sensors or accurately simulate the environment.

REFERENCES

[1] Pulkit Agrawal, Ashvin V Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In Advances in Neural Information Processing Systems, pages 5074–5082, 2016.
[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058, 2017.
[3] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. arXiv preprint arXiv:1808.00177, 2018.
[4] Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. Robotics and Autonomous Systems, 61(1):49–73, 2013.
[5] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski,
Fig. 4: The success (top) and the final distance to goal (bottom) of different methods in different environments. The inputs to the policy are RGBD images rendered in simulation.

(a) Robot setup  (b) Current observation  (c) Goal observation

Fig. 5: Visual Sawyer Reacher setup (left); an example observation image (middle); an example goal image (right).

Fig. 6: Success and final distance to goal on the Visual Sawyer Reacher task shown in Figure 5. Due to the noise incurred in evaluation, we are plotting the average value of a sliding window with a size of 2.5K time steps.

Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In Advances in Neural Information Processing Systems, pages 1471–1479, 2016.

[6] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. arXiv preprint arXiv:1810.05687, 2018.

[7] Marco Cusumano-Towner, Arjun Singh, Stephen Miller, James F O’Brien, and Pieter Abbeel. Bringing clothing into desired configurations with limited perception. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 3893–3900. IEEE, 2011.

[8] Frederik Ebert, Chelsea Finn, Alex X Lee, and Sergey Levine. Self-supervised visual planning with temporal skip connections. arXiv preprint arXiv:1710.05268, 2017.

[9] Frederik Ebert, Sudeep Dasari, Alex X Lee, Sergey Levine, and Chelsea Finn. Robustness via retrying: Closed-loop robotic manipulation with self-supervised learning. arXiv preprint arXiv:1810.03043, 2018.

[10] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. arXiv preprint arXiv:1812.00568, 2018.

[11] Kuan Fang, Yuke Zhu, Animesh Garg, Andrey Kurenkov, Viraj Mehta, Li Fei-Fei, and Silvio Savarese. Learning task-oriented grasping for tool manipulation from simulated self-supervision. arXiv preprint arXiv:1806.09266, 2018.

[12] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In Robotics and Automation (ICRA), 2017 IEEE International Conference on, pages 2786–2793. IEEE, 2017.
[13] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In Advances in neural information processing systems, pages 64–72, 2016.

[14] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 512–519. IEEE, 2016.

[15] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. arXiv preprint arXiv:1709.04905, 2017.

[16] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In 2018 IEEE International Conference on Robotics and Automation (ICRA), 2017 IEEE International Conference on, pages 3389–3396. IEEE, 2017.

[17] Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In Advances in Neural Information Processing Systems, pages 1109–1117, 2016.

[18] Sandy H Huang, Jia Pan, George Mulcaire, and Pieter Abbeel. Leveraging appearance priors in non-rigid registration, with application to manipulation of deformable objects. In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, pages 878–885. IEEE, 2015.

[19] Shervin Javdani, Sameep Tandon, Jie Tang, James F O’Brien, and Pieter Abbeel. Modeling and perception of deformable one-dimensional objects. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 1607–1614. IEEE, 2011.

[20] Leslie Pack Kaelbling. Learning to achieve goals. In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Chambery, France, 1993. Morgan Kaufmann. URL http://people.csail.mit.edu/lpk/papers/ijcai93.ps.

[21] Petar Korushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, pages 3232–3237. IEEE, 2010.

[22] Alex X Lee, Max A Goldstein, Shane T Barratt, and Pieter Abbeel. A non-rigid point and normal registration algorithm with applications to learning from demonstrations. In ICRA, pages 935–942, 2015.

[23] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research, 17(1):1334–1373, 2016.

[24] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[25] Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Parametrized shape models for clothing. In Robotics and Automation (ICRA), 2011 IEEE International Conference on, pages 4861–4868. IEEE, 2011.

[26] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In NeurIPS, pages 9209–9220, 2018.

[27] OpenAI. Learning Dexterous In-Hand Manipulation. pages 1–27, 2018. URL http://arxiv.org/abs/1808.00177.

[28] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In International Conference on Machine Learning (ICML), volume 2017, 2017.

[29] Xue Bin Peng, Angjoo Kanazawa, Sam Toyer, Pieter Abbeel, and Sergey Levine. Variational discriminator bottleneck: Improving imitation learning, inverse rl, and gans by constraining information flow. arXiv preprint arXiv:1810.00821, 2018.

[30] Calder Phillips-Grafflin and Dmitry Berenson. A representation of deformable objects for motion planning with no physical simulation. In Robotics and Automation (ICRA), 2014 IEEE International Conference on, pages 98–105. IEEE, 2014.

[31] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. 06 2018. doi: 10.15607/RSS.2018.XIV.008.

[32] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. arXiv preprint arXiv:1611.04201, 2016.

[33] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In International conference on machine learning, pages 1312–1320, 2015.

[34] Connor Schenck and Dieter Fox. Guided policy search with delayed sensor measurements. arXiv preprint arXiv:1609.03076, 2016.

[35] Jürgen Schmidhuber. Curious model-building control systems. In Neural Networks, 1991. 1991 IEEE International Joint Conference on, pages 1458–1463. IEEE, 1991.

[36] Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). IEEE Transactions on Autonomous Mental Development, 2(3):230–247, 2010.

[37] John Schulman, Alex Lee, Jonathan Ho, and Pieter Abbeel. Tracking deformable objects with point clouds. In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 1130–1137. IEEE, 2013.

[38] Pierre Sermanet, Kelvin Xu, and Sergey Levine. Unsupervised perceptual rewards for imitation learning. arXiv preprint arXiv:1612.06699, 2016.

[39] Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, Stefan Schaal, Sergey Levine, and Google Brain. Time-contrastive networks: Self-supervised learning from video. In 2018 IEEE Interna-
Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots. *arXiv preprint arXiv:1804.10332*, 2018.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 23–30. IEEE, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

Ping Chuan Wang, Stephen Miller, Mario Fritz, Trevor Darrell, and Pieter Abbeel. Perception for the manipulation of socks. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 4877–4884. IEEE, 2011.

David Warde-farley, Tom Van De Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised Control through Non-parameteric Discriminative Rewards. pages 1–17, 2018.

Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 79–86. IEEE, 2017.

Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.
A Environment Details

During evaluation, for all environments, a binary sparse reward is given at each time step. A positive reward \( R_+ = 1 \) is given when the goal is reached, i.e. \( ||s_{t+1} - s_g|| \leq \epsilon \) and a negative reward \( R_- = -1 \) is given otherwise. Other environment details are summarized in Table 1.

| Environment               | Observation Dimension | Goal Dimension | Rendered Dimension | Action Dimension | Horizon (T) | \( \epsilon \) (m) |
|---------------------------|-----------------------|----------------|--------------------|------------------|-------------|-----------------|
| Reacher                   | 10                    | 2              | 100x100x3          | 2                | 50          | 0.01            |
| FetchReach                | 10                    | 3              | 100x100x4          | 3                | 50          | 0.05            |
| FetchPush                 | 25                    | 3              | -                 | 4                | 50          | 0.05            |
| RopePush                  | 45                    | 30             | 100x100x3          | 4                | 3           | 0.1             |
| VisualReacher (Sawyer)    | -                     | -              | 100x100x4          | 3                | 25          | 0.1             |

Table 1: Summarized environment details. The observation and goal dimension are the dimensions of the low dimension state representation when available. The rendered dimensions are the dimensions of the rendered RGBD images used in the visual experiments.

**Sawyer robot experiment details:** The observation is recorded with an Intel RealSense D435 depth camera. The goal observations are sampled by moving the robot arm to a uniformly sampled location in a cuboid of diagonal length 1.3m. The episode was considered successful if the end effector moved to within 0.1 m from goal location at the end of the episode (we used a time horizon of 25 steps). The trained policy performs position control and outputs end effector displacement within a range of -0.05m to 0.05m in each direction.

B Hyper-parameters

All the experiments are run for two random seeds. The hyper-parameters of the training algorithm with indicator rewards are summarized in Table 2. For all experiments with visual observation, the parameters of the convolution layers are shared among the observation input and goal input. Due to the complexity of the RopePush environment, a spatial softmax layer [1] with an output size of 32 is applied before the fully connected layers.

C Ablation Analysis

We show different ablations of our methods in Supplementary Figure 1. We can see that, in the RopePush environment, filtering is required for the policy to learn. On the other hand, the Reacher and FetchReach environments show that balancing is required for optimal performance. In all cases, Indicator+Balance+Filter consistently performs better than all the ablated methods. Thus, these results show that both balancing and filtering are important for optimal performance across a range of tasks.

References

[1] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
| Parameter | Value |
|-----------|-------|
| **shared** | |
| positive reward ($R_+$) | 1 |
| negative reward ($R_-$) | -1 |
| reward filtering ($q_0$) | $\frac{1}{2} \left[ R_- + \gamma R_+/(1 - \gamma) + R_+/(1 - \gamma) \right]$ |
| optimizer | Adam [2] |
| learning rate | 0.001 |
| discount ($\gamma$) | $\frac{T-1}{T}$ |
| target network smoothing ($\tau$) | 0.98 |
| nonlinearity | tanh |
| **state observation** | |
| replay buffer size | $10^6$ |
| minibatch size | 256 |
| network architecture | 3 hidden layers with 256 neurons for each |
| **visual observation** | |
| replay buffer size | $5 \cdot 10^3$ |
| minibatch size | 128 |
| network architecture | 4 convolution layers followed by 3 hidden layers with 256 neurons for each |

Table 2: Summarized hyper-parameters.

Supplementary Figure 1: The success (top) and the final distance to goal (bottom) of different ablations of our method.