

Reinforcement Learning for Robotic Manipulation using Simulated Locomotion Demonstrations

Ozsel Kilinc, Yang Hu, and Giovanni Montana

Abstract—Learning robot manipulation policies through reinforcement learning (RL) with only sparse rewards is still considered a largely unsolved problem. Although learning with human demonstrations can make the training process more sample efficient, the demonstrations are often expensive to obtain, and their benefits heavily depend on the expertise of the demonstrators. In this paper we propose a novel approach for learning complex robot manipulation tasks with self-learned demonstrations. We note that a robot manipulation task can be interpreted, from the object’s perspective, as a locomotion task. In a virtual world, the object might be able to learn how to move from its initial position to the final target position on its own, without being manipulated. Although objects cannot move on their own in the real world, a policy to achieve object locomotion can be learned through physically-realistic simulators, which are nowadays widely available and routinely adopted to train RL systems. The resulting object-level trajectories are called Simulated Locomotion Demonstrations (SLD). The SLDs are then leveraged to learn the robot manipulation policy through deep RL using only sparse rewards. We thoroughly evaluate the proposed approach on 13 tasks of increasing complexity, and demonstrate that our framework can result in faster learning rates and achieve higher success rate compared to alternative algorithms. We demonstrate that SLDs are especially beneficial for complex tasks like multi-object stacking and non-rigid object manipulation.

Index Terms—reinforcement learning, sparse reward, robotic, manipulation, dexterous hand, stacking, non-rigid object

I. INTRODUCTION

Reinforcement Learning (RL) solves sequential decision making problems by learning a policy that maximises expected rewards. Recently, with the aid of deep artificial neural network as function approximators, RL-trained agents are able to master a number of tasks, most notably solving video games [1] and board games [2]. Robot manipulation tasks, on the other hand, are particularly challenging and still partially unsolved due to their complexity. These tasks may involve multiple stages (e.g. stacking multiple blocks), high-dimensional state spaces (e.g. dexterous hand manipulation [3], [4]) and complex dynamics (e.g. manipulating non-rigid objects). Although existing RL algorithms have achieved impressive performance on a wide range of robot manipulation tasks like grasping [5], [6], stacking [7] and dexterous

hand manipulation [3], [4], these algorithms usually require carefully-designed reward signals to learn good policies. For example, [6] propose a thoroughly weighted 5-term reward formula for learning to stack Lego blocks and [8] use a 3-term shaped reward to perform door-opening tasks with a robot arm. However, the requirement of hand-engineered reward functions limits the applicability of RL in real-world robot manipulation to cases where task-specific knowledge can be gathered and captured.

As an alternative to using shaped rewards that embed problem-specific knowledge, learning with a sparse reward (i.e. a binary reward indicating the completion of the task) is practically more desirable as it generalises to many tasks and does not require hand-engineering [2], [9], [10], yet it is very challenging to achieve. Learning with sparse reward requires the RL agent to discover a sequence of actions without any feedback on the their “correctness” until the task is finally accomplished. Existing approaches to address RL with sparse rewards [9]–[16] are able to complete a few robot manipulation tasks like object pushing [9], [16], pick-and-place [9], stacking two blocks [11], [16], and target finding in a scene [14], [15]. Nevertheless, some of the aforementioned tasks, e.g. stacking multiple blocks and manipulating non-rigid objects, still remains challenging.

A particularly promising approach to facilitate learning has been to introduce a human expert providing examples on how to complete a task. The agent can utilise these demonstrations in various ways, e.g. by generating a policy mimicking the demonstrations [17]–[19], pre-learning a policy from demonstrations for further RL [2], [20], utilising the demonstrations to guide exploration [7], learning a reward function from demonstrations [21]–[24], and combining demonstrations and trajectories generated during RL to facilitate policy learning [25]–[27]. Practically, however, human demonstrations are expensive to obtain. Furthermore, the effectiveness of the demonstrations depends on the competence of the demonstrators. Demonstrators with insufficient task-specific knowledge could result in low quality demonstrations and hence sub-optimal policies. Although there is an existing body of work focusing on learning with imperfect demonstrations [28]–[33], these methods usually assume that either qualitative evaluations are available [28], [30], [32] or require a substantial amount of (optimal) demonstrations to achieve good performance [29], [31], [33].

In this paper, we propose a novel approach using artificially-generated demonstrations to enable learning robot manipulation tasks through deep RL with sparse rewards. Our method builds on the following observation. A robot manipulation

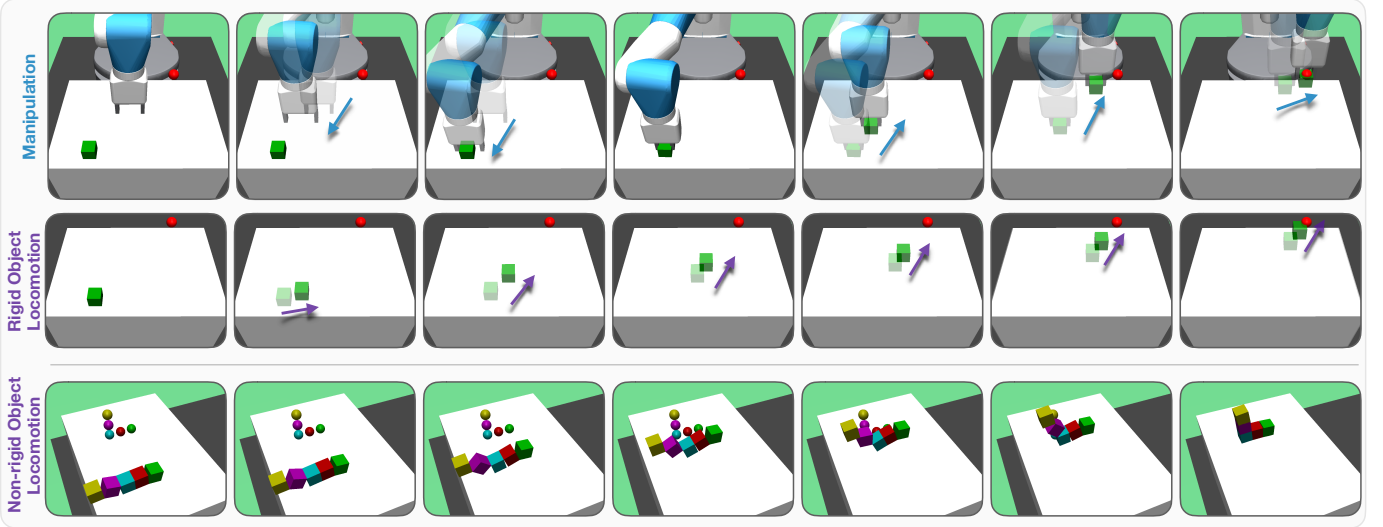


Fig. 1. An illustration of the proposed approach. Top row: a general robot manipulation task of pick-and-place, which requires the robot to pick up an object (green cube) and place it to a specified location (red sphere). Middle row: the corresponding auxiliary locomotion task requires the object to move to the target location. Bottom row: the auxiliary locomotion task corresponding to a pick-and-place task with a non-rigid object (not shown). Note that the auxiliary locomotion tasks usually have significantly simpler dynamics compared to the corresponding robot manipulation task, hence can be learnt efficiently through standard RL, even for very complex tasks. The learnt locomotion policy is used to inform the robot manipulation policy.

task can be viewed as an object locomotion task when seen from the point of view of the object being manipulated. Typically, the robot is required to manipulate each object so that, eventually, the object reaches a target position through a sequence of actions that modify its location and pose. For example, in the pick and place manipulation task illustrated in Fig 1 (top row), the object is picked up by a two-finger gripper and moved from an initial position to a pre-defined target position (red dot). In the associated object locomotion task (middle row), the object needs to move to the target position on its own. More complex tasks, such as manipulating a non-rigid object, can also be associated with a corresponding object locomotion task (bottom row). Although in the real world it is impossible for an object to move on its own, object locomotion can be achieved in a virtual environment by leveraging existing physics engines, which are now commonplace for RL training of complex robotics systems, such as MuJoCo [34], Gazebo [35] and Pybullet [36].

We exploit the above observation as follows. First, we propose to train an object to learn an optimal locomotion policy on its own, without being explicitly manipulated, through a 3D physics simulator. This object-level locomotion policy enables the object to reach a target position from a starting position through a sequence of movements, which may be partially constrained by the presence of the robot. Learning this auxiliary task is usually much simpler than the main manipulation task, since the objects usually operate in simpler state/action spaces and/or have simpler dynamics compared to the manipulating robot. Once the locomotion policy has been learnt, we utilise it to produce object-level demonstrations that inform the robot’s manipulation policy, and call them Simulated Locomotion Demonstrations (SLDs). We utilise SLDs in the form of an auxiliary reward guiding the learning of the manipulation policy. This reward encourages

the robot manipulation to produce similar object locomotion compared to SLDs. Although SLD relies on a simulator, this requirement does not restrict their applicability to real world problems because the manipulation policies can still be learned with physical systems, and the object locomotion policies do not need to be executed. We adopt deep deterministic policy gradient (DDPG) [37] to learn two individual state-value functions for the sparse environmental reward resulting from the main task, and the SLD reward, and simultaneously learn a policy to produce actions maximising the two state-value functions. Here we focus on DDPG due to its widely reported effectiveness for robot manipulation tasks. However, the use of SLDs defines a general framework for which most on-policy and off-policy RL algorithms could also be used.

The proposed methodology has been tested on 13 continuous control environments using the MuJoCo physics engine [34] within the OpenAI Gym framework [38]. Our test environments cover a variety of robot manipulation tasks with increasing level of complexities, e.g. pushing, sliding and pick-and-place tasks with a Fetch robotic arm, in-hand object manipulation with a Shadow’s dexterous hand, multi-object stacking, and non-rigid object manipulation. Overall, across all 13 environments, we have found that our approach can achieve faster learning rate and higher success rate compared to existing methods, especially in complex tasks such as stacking objects and manipulating non-rigid objects.

The remainder of the paper is organised as follows. In Section II, we provide some introductory background material regarding the RL framework used in this work. In Section III, we develop the proposed methodology. In Section IV, we describe all the environments used for our experiments. In Section V, we report the experimental results. Finally, in Section VI, we review all the related work before providing a discussion and suggestions for further extensions in Sec-

tion VII.

II. BACKGROUND

A. Reinforcement Learning

Reinforcement learning (RL) tackles sequential decision making problems with the aim of maximising a reward signal. This is done by learning optimal actions under different situations (i.e. states). Standard RL approaches consider an agent interacting with an environment, modelled by Markov Decision Process (MDP). A MDP is defined by tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where \mathcal{S} is the state space, \mathcal{A} is the action space, \mathcal{T} is dynamics or transition function, \mathcal{R} is the reward function, and $\gamma \in (0, 1)$ the discounting factor. At a time step, the agent receives the state of the environment $s \in \mathcal{S}$, and chooses an action based on a policy $a = \mu(s|\theta_\mu) : \mathcal{S} \times \mathcal{A}$ where θ_μ is the policy parameters. A reward is received for this action, $r = \mathcal{R}(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Then, the environment moves into the next state $s' \in \mathcal{S}$ according to the transition function, i.e. $s' = \mathcal{T}(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$. The aim of RL is to learn the policy to maximise the expected return $J(\theta_\mu) = \mathbb{E}_{s_t, a_t \in \mu} \sum_t \gamma^t \mathcal{R}(s_t, a_t)$ where s_t and a_t are the state and action in time-step t , respectively.

B. Deep Deterministic Policy Gradient Algorithm

Policy Gradient (PG) algorithms maximises the expected return $J(\theta_\mu) = \mathbb{E}_{s_t, a_t \in \mu} \sum_t \gamma^t \mathcal{R}(s_t, a_t)$ by updating θ_μ in the direction of $\nabla_{\theta_\mu} J(\theta_\mu)$. Deep Deterministic Policy Gradient (DDPG) learns deterministic policies [39] modelled by Deep Neural Networks (DNNs) to leverage the state-of-the-art representation capacity of DNNs.

Concretely, DDPG includes a policy (actor) network $a = \mu(s|\theta_\mu)$ and an action-value (critic) network $Q^\mu(s, a|\theta_Q)$ where θ_Q contains the critic's parameters. The actor maps states to deterministic actions. The critic estimates the total expected return starting from the state s and taking action a . DDPG alternates between two stages. First, it collects experience using the current policy with an additional noise sampled from a random process \mathcal{N} for exploration, i.e. $a = \mu(s|\theta_\mu) + \mathcal{N}$. The algorithm stores the experienced transitions, i.e. $\langle s, a, r, s' \rangle$, into a replay buffer \mathcal{D} . Then, it learns the actor and critic networks simultaneously using the experienced transitions in \mathcal{D} . The critic network is learnt by minimising the following loss to satisfy the Bellman equation, similarly to Q-learning [40]:

$$\mathcal{L}(\theta_{Q^\mu}) = \mathbb{E}_{s, a, r, s' \sim \mathcal{D}} \left[(Q^\mu(s, a|\theta_{Q^\mu}) - y)^2 \right] \quad (1)$$

where $y = r + \gamma Q^\mu(s', \mu(s')|\theta_{Q^\mu})$ is the target value. Practically, directly minimising Eq. 1 is proven to be unstable, since the function Q^μ used in target value estimation is also updated in each iteration. Similarly to DQN [1], DDPG stabilises the learning by obtaining smoother target values $y = r + \gamma Q^{\mu'}(s', \mu'(s')|\theta_{Q^{\mu'}})$, where μ' and $Q^{\mu'}$ are target networks. The weights of μ' and $Q^{\mu'}$ are the exponential moving average of the weights of μ and Q^μ over iterations, respectively. The actor, on the other hand, is updated using the

following policy gradient, in order to take actions maximising the expected return:

$$\nabla_{\theta_\mu} J(\theta_\mu) = \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_a Q^\mu(s, a|\theta_{Q^\mu})|_{a=\mu(s)} \nabla_{\theta_\mu} \mu(a|\theta_\mu) \right] \quad (2)$$

C. Hindsight Experience Replay

Hindsight Experience Replay (HER) [9] has been introduced to learn policies from sparse rewards for robot manipulation tasks. The idea is to view the states achieved in an episode as pseudo goals (i.e. achieved goals) to facilitate learning, even when this episode fails to achieve the desired goal. Specifically, let $\langle s||g, a, r, s' ||g \rangle$ be the original transition obtained in an episode, where $||$ denotes the concatenation and g the desired goal of the task. The original environment will not reward the agent if g is not achieved. Learning the policy under these conditions is difficult as no reward is obtained if g is not achieved. In HER, g is replaced by an achieved goal denoted by g' , which is randomly sampled from the states reached in an episode. This generates new transition $\langle s||g', a, r, s' ||g' \rangle$ which is more likely to be rewarded. The generated transitions are saved into an experience replay buffer and can be used for off-policy RL algorithms like DQN [1] and DDPG [37]. HER allows the agent to start learning from goals that are more easily achieved, and ultimately learns the desired goal when $g' \rightarrow g$.

III. METHODOLOGY

A. Problem Formulation

We model robot manipulation tasks as MDPs $G = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$. Our aim is to learn a manipulation policy for the robot that maximises the expected return $J(\theta_\mu)$. We follow the notation introduced in Section II-A, and consider a sparse reward setting, where the robot receives a binary reward indicating the completion of the task. Furthermore, for a given task modelled by G , we introduce an auxiliary object locomotion task through a paired MDP, $G_{obj} = \langle \mathcal{S}_{obj}, \mathcal{A}_{obj}, \mathcal{T}_{obj}, \mathcal{R}_{obj}, \gamma_{obj} \rangle$. For $s_{obj} \in \mathcal{S}_{obj}$, we use $s_{obj} = \psi(s)$, where ψ is an operator that extracts object-related states (see Section IV). This is because robot-related states are not necessary for the object to accomplish the locomotion task. The aim of this auxiliary task is to learn a locomotion policy for the object, $a_{obj} = \mu_{obj}(s_{obj})$, that maximises the expected return. Given a manipulation task, we learn μ_{obj} firstly, due to its simplicity in most tasks, and then utilise μ_{obj} to guide μ , which is typically much more complex.

B. Simulated Locomotion Demonstrations (SLD)

In most robot manipulation tasks, the object locomotion is caused by the actions taken by the robot, which are driven by μ . Let a'_{obj} be the object locomotion caused by manipulation policy. a'_{obj} can be expressed as a function of μ : $a'_{obj} = f(\mu(s|\theta_\mu)) = f(a)$. Assuming we have learnt $a_{obj} = \mu_{obj}(s_{obj}) = \mu_{obj}(\psi(s))$, we utilise a_{obj} as a demonstration of desired object locomotion. We make two rather weak assumptions to utilise a_{obj} to guide the learning of μ .

Assumption 1. $a'_{obj} \in \mathcal{A}_{obj}$.

Assumption 2. For optimal μ and μ_{obj} , we have $P(s_{obj}|\mu) \stackrel{d}{=} P(s_{obj}|\mu_{obj})$ where $P(s_{obj}|\mu)$ and $P(s_{obj}|\mu_{obj})$ are the distribution of s_{obj} under μ and μ_{obj} , respectively. The parameter θ_μ could be learnt by minimising the Euclidean distance between a'_{obj} and a_{obj} , i.e.

$$\begin{aligned} & \arg \min_{\theta_\mu} \mathbb{E}_{s \sim P(s|\mu)} \left\| a'_{obj} - a_{obj} \right\|_2^2 \\ & = \arg \min_{\theta_\mu} \mathbb{E}_{s \sim P(s|\mu)} \left\| f(\mu(s|\theta_\mu)) - \mu_{obj}(\psi(s)) \right\|_2^2 \end{aligned} \quad (3)$$

where $P(s|\mu)$ is the state distribution under policy μ . Note that Assumption 1 ensures that $a'_{obj} = f(\mu(s|\theta_\mu)) = f(a)$ is in the same action space as a_{obj} , so that a_{obj} can be used as a demonstration to learn μ , whereas Assumption 2 ensures that μ_{obj} can be applied to $\psi(s)$ when $s \sim P(s|\mu)$.

Eq. 3 encourages the object locomotion generated by the robot to be sufficiently close to the desired one. Practically, however, we do not have a closed-form expression for the required mapping, f . This is because the robot could have a complex architecture, and the objects could have various shapes, resulting in very complex dynamics when the robot interacts with the objects. Rather than assuming knowledge of f to calculate a'_{obj} , we estimate an inverse dynamic model that predicts a'_{obj} given the states occurring before and after an action. Specifically, a robot state s corresponds to an object state s_{obj} . Upon taking an action $a = \mu(s|\theta_\mu)$, a new robot state s' is observed, which corresponds to a new object state s'_{obj} . We learn a function approximator through an artificial neural network, \mathcal{I} , to estimate a'_{obj} from s_{obj} and s'_{obj} , i.e. $a'_{obj} \approx \mathcal{I}(s_{obj}, s'_{obj}|\phi)$ where ϕ is the parameters of \mathcal{I} . Since the objects usually have much simpler dynamics compared to the robot, typically \mathcal{I} is much easier to learn compared to f . We learn \mathcal{I} together with the locomotion policy (see Section III-C and Alg. 1). Substituting \mathcal{I} into Eq. 3 leads to the following optimisation problem:

$$\arg \min_{\theta_\mu} \mathbb{E}_{s \sim P(s|\mu)} \left\| \mathcal{I}(s_{obj}, s'_{obj}|\phi) - \mu_{obj}(s_{obj}) \right\|_2^2 \quad (4)$$

Note that s'_{obj} is a function of μ : $s'_{obj} = \psi(s')$ where $s' = \mathcal{T}(s, a) = \mathcal{T}(s, \mu(s|\theta_\mu))$.

Minimising Eq. 4 through gradient-based methods is difficult as it requires differentiation through \mathcal{T} , which is unknown. Nevertheless, since both \mathcal{I} and μ_{obj} are known, one option to minimise Eq. 4 would be to learn a manipulation policy μ through standard model-free RL using the following reward:

$$q = - \left\| \mathcal{I}(s_{obj}, s'_{obj}) - \mu_{obj}(s_{obj}) \right\|_2^2 \quad (5)$$

Practically, however, the above reward is sensitive to the scale of object actions, i.e. as the object moves close to the target, the scale of $\mu_{obj}(s_{obj})$ decreases significantly, which makes the reward too weak for learning. In order to deal with this issue, we introduce $Q^{obj}(s_{obj}, a_{obj})$, the expected return when the action a_{obj} is taken at the state s_{obj} (i.e. RL action-value function), and use it to build a reward in analogous to Eq. 5. Note that $Q^{obj}(s_{obj}, a_{obj})$ is well-bounded [41] and its scale increases as the object moves nearer to the target in

Algorithm 1: Learning locomotion policy and inverse dynamic

Given : Locomotion MDP $G_{obj} = \langle S_{obj}, \mathcal{A}_{obj}, \mathcal{T}_{obj}, \mathcal{R}_{obj}, \gamma_{obj} \rangle$,
Neural networks $\mu_{obj}(\cdot|\theta_{\mu_{obj}})$, $Q^{\mu_{obj}}(\cdot|\theta_{Q^{\mu_{obj}}})$ and $\mathcal{I}(\cdot|\phi)$
A random process \mathcal{N}_{obj} for exploration
Initialise: Parameters $\theta_{\mu_{obj}}$, $\theta_{Q^{\mu_{obj}}}$ and ϕ
Experience replay buffer \mathcal{D}_{obj}
for $i_{episode} = 1$ **to** $N_{episode}$ **do**
 for $i_{rollout} = 1$ **to** $N_{rollout}$ **do**
 Sample an initial state s
 for $t = 0, T-1$ **do**
 Sample an object action: $a_{obj} = \mu_{obj}(s_{obj}) + \mathcal{N}_{obj}$
 Execute the action:
 $s'_{obj} = \mathcal{T}_{obj}(s_{obj}, a_{obj})$, $r_{obj} = \mathcal{R}_{obj}(s_{obj}, a_{obj})$
 Store $(s_{obj}, a_{obj}, r_{obj}, s'_{obj})$ in \mathcal{D}_{obj}
 $s_{obj} \leftarrow s'_{obj}$
 Generate HER samples and store in \mathcal{D}_{obj}
 for $i_{update} \leftarrow 1$ **to** N_{update} **do**
 Get a random mini-batch of samples from \mathcal{D}_{obj}
 Update $Q^{\mu_{obj}}$ minimising the loss in Eq.(1)
 Update μ_{obj} using the gradient in Eq.(II-B)
 Update \mathcal{I} minimising the loss in Eq.(7)
Return : μ_{obj} , $Q^{\mu_{obj}}$ and \mathcal{I}

sparse reward setting. $Q^{obj}(s_{obj}, a_{obj})$ can be obtained when learning the object policy (see Section III-C). The new reward is written as follows:

$$q = Q^{\mu_{obj}}(s_{obj}, \mathcal{I}(s_{obj}, s'_{obj})) - Q^{\mu_{obj}}(s_{obj}, \mu_{obj}(s_{obj})) \quad (6)$$

We refer to Eq. 6 as the SLD reward. Since μ_{obj} is learnt through standard RL, $a_{obj} = \mu_{obj}(s_{obj})$ can be viewed as the desired action maximising the expected return when the object is at state s_{obj} . Accordingly, Eq. 6 can be viewed as the advantage of the object action caused by the manipulation policy $a'_{obj} \approx \mathcal{I}(s_{obj}, s'_{obj})$ with respect to the desired object action $a_{obj} = \mu_{obj}(s_{obj})$. Maximising this term encourages the robot manipulation to produce similar object actions compared to the desired ones.

C. Learning Algorithm

In this subsection, we detail the algorithm to learn locomotion policy, and the algorithm to learn the manipulation policy with SLD reward.

Locomotion policy. Alg. 1 describes the learning of locomotion policy. To learn the policy using only sparse rewards, we adopt HER to generate transition samples as in Section II-C. We use DDPG to learn the policy μ_{obj} and the action-value function Q^{obj} , i.e. using the gradient in Eq. II-B to update μ_{obj} and minimizing Eq. 1 to update Q^{obj} . Concurrently, we learn the inverse dynamics model \mathcal{I} using the trajectories generated during the policy learning process by minimising the following objective function:

$$\arg \min_{\phi} \mathbb{E}_{s_{obj}, a_{obj}, s'_{obj} \sim \mathcal{D}_{obj}} \left\| \mathcal{I}(s_{obj}, s'_{obj}|\phi) - a_{obj} \right\|_2^2 \quad (7)$$

where \mathcal{D}_{obj} is an experience replay buffer as in Alg. 1. Recall that $s'_{obj} = \mathcal{T}_{obj}(s_{obj}, a_{obj}) = \mathcal{T}_{obj}(s_{obj}, \mu_{obj}(s_{obj}))$.

Manipulation policy. Alg. 2 describes the learning procedure for the manipulation policy μ . In addition to the action-value function for the robot $Q_r^\mu(s, a)$, we learn another action-

Algorithm 2: Learning manipulation policy

Given : Manipulation MDP $G = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$,
 Learnt object-related components μ_{obj} , $Q^{\mu_{obj}}$ and $\mathcal{I}(\cdot|\phi)$
 Neural networks $\mu(\cdot|\theta_\mu)$, $Q_r^\mu(\cdot|\theta_{Q_r^\mu})$ and $Q_q^\mu(\cdot|\theta_{Q_q^\mu})$
 A random process \mathcal{N} for exploration
 A function ψ extracting object-related states from s
Initialise: Parameters θ_μ , $\theta_{Q_r^\mu}$ and $\theta_{Q_q^\mu}$
 Experience replay buffer \mathcal{D}

for $i_{episode} = 1$ **to** $N_{episode}$ **do**
 for $i_{rollout} = 1$ **to** $N_{rollout}$ **do**
 Sample an initial state s
 for $t = 0, T-1$ **do**
 Sample a robot action: $a = \mu(s) + \mathcal{N}$
 Execute the action:
 $s' = \mathcal{T}(s, a)$, $r = \mathcal{R}(s, a)$
 Obtain SLD reward:
 $s_{obj} = \psi(s)$ and $s'_{obj} = \psi(s')$
 $q = Q^{\mu_{obj}}(s_{obj}, \mathcal{I}_\phi(s_{obj}, s'_{obj})) - Q^{\mu_{obj}}(s_{obj}, \mu_{obj}(s_{obj}))$
 Store (s, a, r, q, s') in \mathcal{D}
 $s \leftarrow s'$
 Generate HER samples and store in \mathcal{D}
 for $i_{update} \leftarrow 1$ **to** N_{update} **do**
 Get a random mini-batch of samples from \mathcal{D}
 Update Q_r^μ minimising the loss in Eq.(1) for r
 Update Q_q^μ minimising the loss in Eq.(1) for q
 Update μ using the gradient in Eq.(8)

Return : μ

value function $Q_q^\mu(s, a)$ for the SLD reward in Eq. 6. The policy parameter θ_μ is updated with the following gradient:

$$\nabla_{\theta_\mu} J(\theta_\mu) = \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_a \left(Q_r^\mu(s, a) + Q_q^\mu(s, a) \right) \Big|_{a=\mu(s)} \nabla_{\theta_\mu} \mu(s|\theta_\mu) \right] \quad (8)$$

Some tasks may include $N > 1$ objects, e.g. stacking. The proposed method is able to handle these tasks by learning individual policies and SLD reward action-value functions $Q_{q_i}^\mu(s, a)$ for each object, $i = 1, \dots, N$. The gradient to update θ_μ is given by

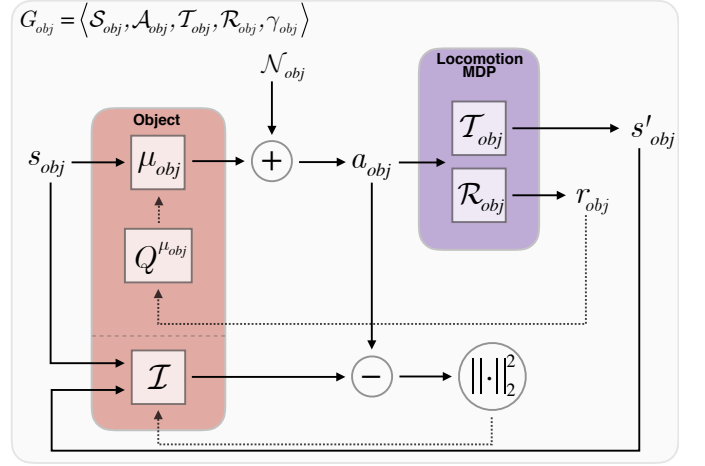
$$\nabla_{\theta_\mu} J(\theta_\mu) = \mathbb{E}_{s \sim \mathcal{D}} \left[\nabla_a \left(Q_r^\mu(s, a) + \sum_{i=1}^N Q_{q_i}^\mu(s, a) \right) \Big|_{a=\mu(s)} \nabla_{\theta_\mu} \mu(s|\theta_\mu) \right] \quad (9)$$

Fig. 2(b) shows the block diagram of manipulation policy learning procedure.

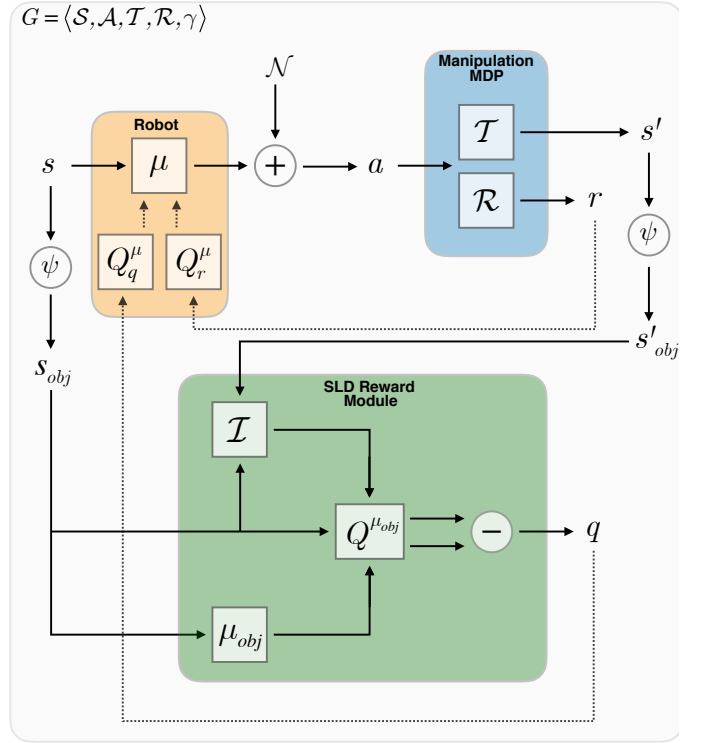
IV. ENVIRONMENTS

We evaluate our method on 13 simulated MuJoCo [34] environments using two different robot configurations: 7-DoF Fetch robotic arm with a two-finger parallel gripper and 24-DoF Shadow's Dexterous Hand. The tasks include single rigid object manipulation, multiple rigid object stacking and non-rigid object manipulation. We adopt 9 MuJoCo environments (3 with Fetch robot arm and 6 with Shadow's hand) used in [42] for single rigid object tasks. Furthermore, we introduce additional environments for stacking multiple object and non-rigid object manipulation using the Fetch robot arm. In all environments the rewards are sparse.

Fetch Arm Single Object Environments. These are the same *Push*, *Slide* and *PickAndPlace* tasks introduced in [42].



(a) Learning locomotion policy μ_{obj} and inverse dynamic \mathcal{I}



(b) Learning manipulation policy μ

Fig. 2. Block diagram for the proposed SLD algorithm. The solid lines represent the forward pass of the model. The dashed lines represent the rewards/losses as feedback signals for model learning.

In each episode, a desired 3D position (i.e. the target) of the object is randomly generated. The reward is 0 if the object is within 5cm range to the target, otherwise the reward is -1 . The robot actions are 4-dimensional: 3D for the desired arm movement in Cartesian coordinates and 1D to control the opening of the gripper. In pushing and sliding, the gripper is locked to prevent grasping. The observations include the positions and linear velocities of the robot arm and the gripper, the object's position, rotation, angular velocity, the object's relative position and linear velocity to the gripper, and the target coordinate. An episode terminates after 50 time-steps.

Shadow’s Hand Single Object Environments. These include the tasks first introduced in [42], i.e. *Egg*, *Block*, *Pen* manipulation. In these tasks, the object (a block, an egg-shaped object, or a pen) is placed on the palm of the robot hand; the robot hand is required to manipulate the object to reach a target pose. The target pose is 7-dimension describing the 3D position together with 4D quaternion orientation, and is randomly generated in each episode. The reward is 0 if the object is within some task-specific range to the target, otherwise the reward is -1 . As in [42], each task has two variants: *Full* and *Rotate*. In the *Full* variant, the object’s whole 7D pose is required to meet the given target pose. In the *Rotate* variants, the 3D object position is ignored and only the 4D object rotation is expected to satisfy the desired target. Robot actions are 20-dimensional controlling the absolute positions of all non-coupled joints of the hand. The observations include the positions and velocities of all 24 joints of the robot hand, the object’s position and rotation, the object’s linear and angular velocities, and the target pose. An episode terminates after 100 time-steps.

Fetch Arm Multiple Object Stacking Environments. The task of stacking multiple objects is built upon the *PickAndPlace* task, taken from the fetch arm single object environments. We consider stacking both two and three objects. For the N -object stacking task, the target has $3N$ dimensions describing the desired positions of all N objects in 3D. Following [7], we start these tasks with the first object placed at its desired target. The robot needs to perform $N - 1$ pick-and-place actions without displacing the first object. The reward is 0 if all objects are within 5cm range to their designated targets, otherwise the reward is -1 . The robot actions and observations are similar to those in the *PickAndPlace* task, excepting that the observations include the position, rotation, angular velocity, relative position and linear velocity to the gripper for each object. The episode length is 50 time-steps for 2 objects and 100 for 3 objects.

Fetch Arm Non-rigid Object Environments. We build non-rigid object manipulation tasks based on the *PickAndPlace* task from the fetch arm single object environments. Instead of using the original rigid block, we have created a non-rigid object by hinging some blocks side-by-side along their edges as shown in Fig. 3. A hinge joint is placed between two neighbouring blocks, allowing one rotational degree of freedom (DoF) along their coincident edges up to 180° . We introduce two different variants: *3-tuple* and *5-tuple*. For the N -tuple task, N cubical blocks are connected with $N - 1$ hinge joints creating $N - 1$ internal DoF. The target pose has $3N$ -dimension describing the desired 3D positions of all N blocks, which are selected uniformly in each episode from a set of predefined target poses (see Fig. 3). The robot is required to manipulate the object to match the target pose. This reward is 0 if all N blocks are within 2cm range to the their corresponding targets, otherwise the reward is -1 . Robot actions and observations are similar to those in the *PickAndPlace* tasks, excepting that the observations include the position, rotation, angular velocity, relative position and linear velocity to the gripper for each block. The episode length is 50 time-steps for both *3-tuple* and *5-tuple*.

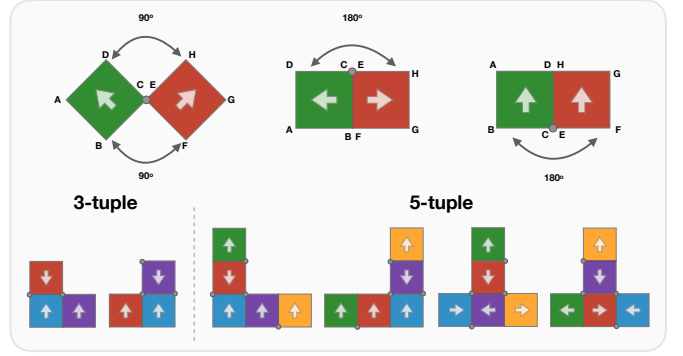


Fig. 3. An illustration of the non-rigid objects used in the experiments. Top row: a hinge joint (shown as grey circles) between two neighbouring blocks allows one rotational DoF along their coincident edges up to 180° . Bottom row: each variant has some predefined target poses (2 options for *3-tuple* and 4 for *5-tuple*).

Object locomotion MDP. As in Section III-A, the proposed method requires to define an object locomotion MDP corresponding to each robot manipulation task. For the environments with the Fetch arm, the object’s observations include the object’s position, rotation, angular velocity, the object’s relative position and linear velocity to the target, and the target location. For the environments with the Shadow’s hand, the object observations include the object’s position and rotation, the object’s linear and angular velocities, and the target pose. The rewards are the same as those in each robot manipulation task. For rigid objects, we define the object action as the desired relative change in the 7D object pose (3D position and 4D quaternion orientation) between two consecutive time-steps. This leads to 7D action spaces. For non-rigid objects, the action includes the desired changes in 7D poses for the blocks at two ends. This leads to 14D action spaces.

It is worth noting that, in the *Full* variants of Shadow’s hand environments, we consider the object translation and rotation as two individual locomotion tasks, and we learn separate locomotion policies and Q-functions for each task. We find that the above strategy encourages the manipulation policy to perform translation and rotation simultaneously. Although object translation and rotation could be executed within a single task, we have empirically found that the resulting manipulation policies tend to prioritise one action versus the other (e.g. they tend to rotate the object first, then translate it) and generally achieves a lower performance.

V. EXPERIMENTS

A. Implementation and Training Process

Three-layer neural networks with ReLU activations were used to approximate all policies, action-value functions and inverse dynamics models. The Adam optimiser [43] was employed to train all the neural networks. During the training of locomotion policies, the robot was always left in the scene and was allowed to occlude with an object, but had otherwise a passive role. In preliminary experiments, we assessed whether letting the robot perform either random or predefined actions whilst the object is training bears any effect on final

performance, and concluded that no particular setting had clear advantages. For learning locomotion and manipulation policies, most of the hyperparameters suggested in the original HER implementation [42] were retained with only a couple of exceptions for locomotion policies only: to facilitate exploration, with probability 0.2 (0.3 in [42]) a random action was drawn from a uniform distribution, otherwise we retained the current action, and added Gaussian noise with zero mean and 0.05 (0.2 in [42]) standard deviation. For locomotion policies, in all Shadow’s hand environments and *5-tuple*, we train the objects over 50 epochs. In the remaining environments, we stop the training after 20 epochs. When training the main manipulation policies, the number of epochs varies across tasks. For both locomotion and manipulation policies, each epoch includes 50 cycles, and each cycle includes 38 rollouts generated in parallel through 38 MPI workers using CPU cores. This leads to $38 \times 50 = 1900$ full episodes per epoch. For each epoch, the parameters are updated 40 times using a batch size of 4864 on a GPU core. We normalise the observations to have zero mean and unit standard deviation as input of neural networks. We update mean and standard deviations of the observations using running estimation on the data in each rollout. We clip the SLD reward to the same range with the environmental sparse rewards, i.e. $[-1, 0]$.

Our algorithm has been implemented in PyTorch¹. All the environments are based on OpenAI Gym, and are provided with support for locomotion policy learning using the *mocap* entity in the MuJoCo library. The corresponding source code, the environments, and illustrative videos for selected tasks have been made publicly available.^{2,3,4}

B. Comparison and Performance Evaluation

We include the following methods for comparisons:

- **DDPG-Sparse:** Refers to DDPG [37] using sparse rewards.
- **HER-Sparse:** Refers to DDPG with HER [9] using sparse rewards.
- **HER-Dense:** Refers to DDPG with HER, using dense distance-based rewards.
- **DDPG-Sparse+SLDR:** Refers to DDPG using sparse environmental rewards and SLD reward proposed in this paper.
- **HER-Sparse+RNDR:** Refers to DDPG with HER, using sparse environmental rewards and random network distillation-based auxiliary rewards (RNDR) [10].
- **HER-Sparse+SLDR:** Refers to DDPG with HER, using sparse environmental rewards and SLD reward.

We use DDPG-Sparse, HER-Sparse and HER-Dense as baselines. HER-Sparse+RNDR is a representative method constructing auxiliary rewards to facilitate policy learning. DDPG-Sparse+SLDR and HER-Sparse+SLDR represents the proposed approach using SLD reward with different methods for policy learning.

Following [42], we evaluate the performance after each training epoch by performing 10 deterministic test rollouts for each one of the 38 MPI workers. Then we compute the test success rate by averaging across the 380 test rollouts. For all comparison methods, we evaluate the performance with 5 different random seeds and report the median test success rate with the interquartile range. In all environments, we also keep the models with the highest test success rate for different methods and compare their performance.

C. Single Rigid Object Environments

The learning curves for Fetch, the *Rotate* and *Full* variants of Shadow’s hand environments are reported in Fig. 4(a), Fig. 4(b) and Fig. 4(c), respectively. We find that HER-Sparse+SLDR features a faster learning rate and the best performance on all the tasks. This evidence demonstrates that SLD, coupled with DDPG and HER, can facilitate policy learning with sparse rewards. The benefits introduced by HER-Sparse+SLDR are particularly evident in hand manipulation tasks (Fig. 4(b) and Fig. 4(c)) compared to fetch robot tasks (Fig. 4(a)), which are notoriously more complex to solve. Additionally, we find that HER-Sparse+SLDR outperforms HER-Sparse+RNDR in most tasks. A possible reason for this result is that most methods using auxiliary rewards are based on the notion of curiosity, whereby reaching unseen states is a preferable strategy, which is less suitable for manipulation tasks [9]. In contrast, the proposed method exploits a notion of desired object locomotion to guide the main policy during training. We also observe that DDPG-Sparse+SLDR fails for most tasks. A possible reason for this is that, despite its effectiveness, the proposed approach still requires a suitable RL algorithm to learn from SLD rewards together with sparse environmental rewards. DDPG on its own is less effective for this task. Finally, We find that HER-Dense performs worse than HER-Sparse. This result support previous observations that sparse rewards may be more beneficial for complex robot manipulation tasks compared to dense rewards [9], [42].

D. Fetch Arm Multiple Object Environments

For environments with N objects, we reuse the locomotion policies trained on the *PickAndPlace* task with single objects, and obtain an individual SLD reward for each of N objects. We train $N+1$ action-value functions in total, i.e. one for each SLD reward and one for the environmental sparse rewards. The manipulation policy is trained using the gradient in Eq. III-C.

Inspired by [42], we randomly select between two initialisation settings for the training: (1) the targets are distributed on the table (i.e. an auxiliary task) and (2) the targets are stacked on top of each other (i.e. the original stacking task). Each initialisation setting is randomly selected with a probability of 0.5. We have observed that this initialisation strategy helps HER-based methods complete the stacking tasks. From Fig. 4(d), we find that HER-Sparse+SLDR achieves better performance compared to HER-Sparse, HER-Sparse+RND and HER-Dense in the 2-object stacking task (Stack2), while other methods fail. On the more complex 3-object stacking task (Stack3), HER-Sparse+SLDR is the only algorithm to

¹<https://pytorch.org/>

²Source code: <https://github.com/WMGDataScience/sldr.git>

³Environments: https://github.com/WMGDataScience/gym_wmgds.git

⁴Supplementary videos: <https://youtu.be/jubZ0dPV12M>

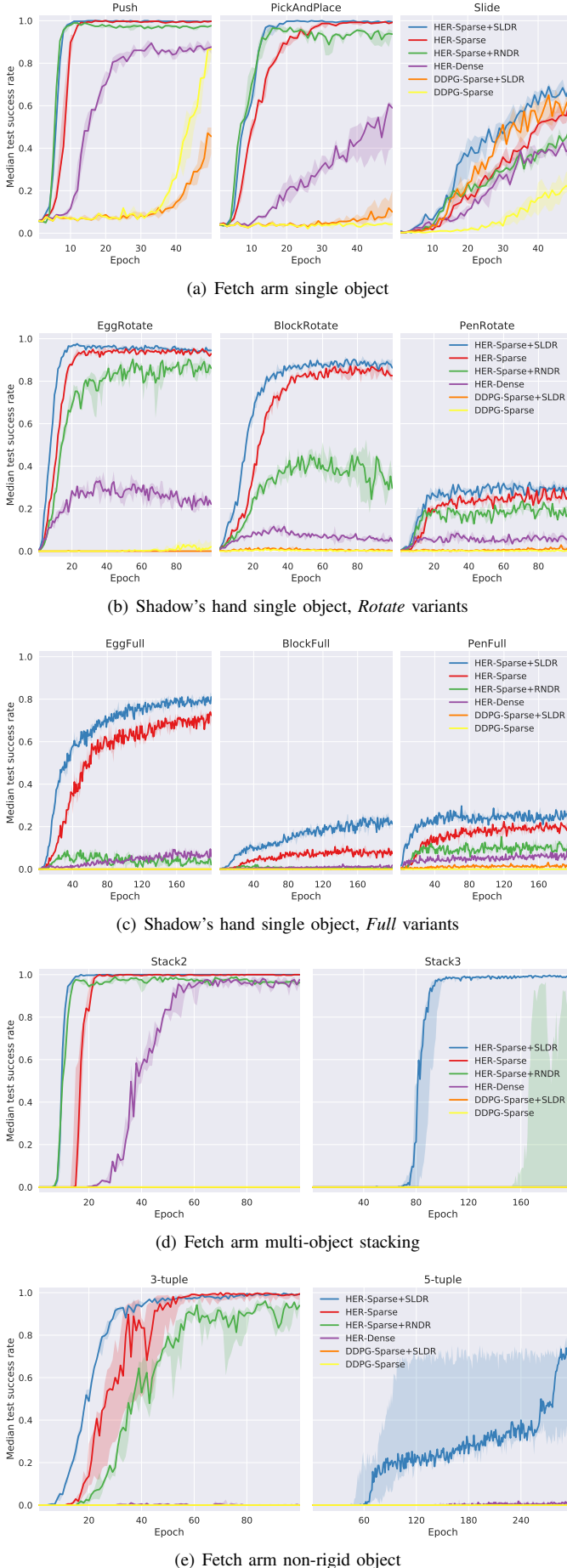


Fig. 4. Learning curves of comparison algorithms on all environments.

succeed. HER-Sparse+RND occasionally solves the Stack3 task with fixed random seeds, but the performance is unstable across different random seeds and multiple runs.

E. Fetch Arm Non-Rigid Object Environments

The learning curves for 3-tuple and 5-tuple non-rigid object tasks are reported in Fig. 4(e). Similarly to the multiple object environment, HER-Sparse+SLDR achieves better performance for the 3-tuple task compared to HER-Sparse and HER-Sparse+RND, while the other methods fail to complete the task. For the more complex 5-tuple task, only HER-Sparse+SLDR is able to succeed. Among the 4 pre-defined targets depicted in Fig. 3, HER-Sparse+SLDR can achieve 3 targets on average, and can accomplish all 4 targets in one instance, out of 5 runs with different random seeds.

F. Comparison Across the Best Models

Fig. 5 summarises the performance of the models with the best test success rates for each one of the competing methods. We can see that the proposed HER-Sparse+SLDR achieves top performance compared to all other methods. Specifically, HER-Sparse+SLDR is the only algorithm that is able to steadily solve 3-object stacking (Stack3) and 5-tuple non-rigid object manipulation (5-tuple). Remarkably, these two tasks have the highest complexity among all the 13 tasks. The Stack3 task includes multiple stages that require the robot to pick and place multiple objects with different source and target locations in a fixed order; in the 5-tuple task the object has the most complex dynamics. For these complex tasks, the proposed SLD reward seems to be particularly beneficial. A possible reason is that, although the task is very complex, the objects are still able to learn good locomotion policies (see Fig 6(a)). The SLD from learnt locomotion policies provides critical information on how the object should be manipulated to complete the task, and this information is not utilised by other methods like HER and HER+RND. Our approach outperforms the runner-up by a large margin in the *Full* variants of Shadow's hand manipulation tasks (EggFull, BlockFull and PenFull), which feature complex state/action spaces and system dynamics. Finally, the proposed method consistently achieves better or similar performance than the runner-up in other simpler tasks.

VI. RELATED WORK

A. Learning from Demonstration

A substantial body of work exists on reinforcement learning with demonstrations. Behaviour cloning (BC) methods approach sequential decision making as a supervised learning problem [18], [19], [44], [45]. Some BC methods include an expert demonstrator in the training loop to handle the mismatching between the demonstration data and the data encountered in the training procedure [17], [46]. Recent BC methods have also considered adversarial frameworks [24], [47] to improve the policy learning. As a different approach, inverse reinforcement learning seeks a reward/cost function to guide policy learning [21]–[23], and several methods have

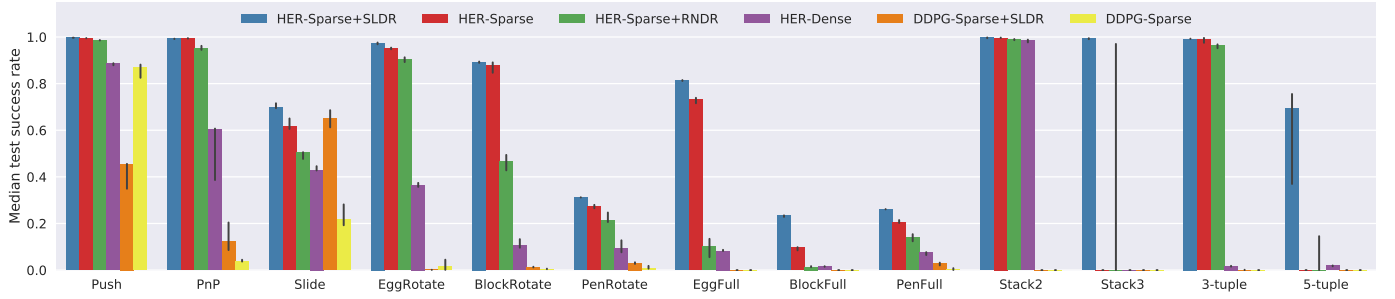


Fig. 5. Comparison of models with the best test success rate for all methods on all the environments.

been developed to leverage demonstrations for robotic manipulation tasks with sparse rewards. [7], [25] mixes demonstration data with trajectories collected during the RL process to guide the exploration. [20] use the demonstrations to pre-learn a policy, which is further fine-tuned in a following RL stage. All the above methods require human demonstrations on the full state and action spaces of the task. In contrast, the proposed method generates demonstrations artificially, and only requires partial demonstrations on object-related state and actions.

B. Goal Conditioned Policy

Goal-conditioned policies [48] that can effectively generalise over multiple goals have shown to be promising for robotic problems. For manipulation tasks with sparse rewards, several approaches have recently been proposed to automatically generate a curriculum of goals to facilitate policy learning: [49] follow a self-play approach on reversible or resettable environments; [50] employ adversarial training for robotic locomotion tasks; [51] use variational autoencoders for visual robotics tasks; [9] introduced Hindsight Experience Replay (HER), which randomly draws synthetic goals from previously encountered states. The automatic goal generation in the above method could be problematic for tasks with multiple stages, e.g. stacking multiple objects. On the other hand, [11] form a curriculum by manually defining 15 semantically grounded sub-tasks each with an individual reward. This method requires human effort to decompose the final task into sub-tasks, which limits its applicability.

C. Auxiliary Reward in RL

Lately, increasing efforts are being spent on designing auxiliary rewards to facilitate learning in tasks with sparse rewards. Many strategies are designed around the notion of curiosity [52], which encourages agents to visit novel states that have not been seen in previous experience. Some of these recent methods are in order. [14] formulate the auxiliary reward using the error in predicting the RL agent’s actions by an inverse dynamics model. [12] encourage the agent to visit the states that result the largest information gain in system dynamics; [10] construct the auxiliary reward based on the error in predicting the output of a fixed randomly initialised neural network, and [15] introduces the notion of state reachability. It has been previously observed that

visiting unseen states may be less beneficial in complex robot manipulation tasks as exploring complex state spaces to find rewards is rather impractical [9].

VII. CONCLUSION AND DISCUSSION

In this paper, we address the problem of mastering robot manipulation through deep reinforcement learning using only sparse rewards. The rationale for the proposed methodology is that robot manipulation tasks can be re-interpreted as involving locomotion tasks when seen from the perspective of the objects being manipulated. Based on this observation, we propose to firstly learn locomotion policies for objects individually through standard RL algorithms, which is achieved through physically-realistic 3D simulators, and then leverage these policies to improve the manipulation learning phase.

We believe that using SLDs introduces three significant advantages over the use of human demonstrations. First, SLDs are generated artificially through a RL policy, hence require no human effort. Second, producing human demonstrations for complex tasks may be difficult to achieve and requires a significant investment in human resources. For instance, it may be particularly difficult for a human to generate good demonstrations for tasks such as manipulating non-rigid objects with a single hand or with a robotic gripper. On the other hand, the locomotion policies can be easily learnt for objects, even for complex tasks, purely in a virtual environment. In our studies, these policies have achieved 100% success rate on all tasks (e.g. see Fig. 6(a) and Fig. 6(b)). Third, since the locomotion policy is learnt through RL, SLD does not require task-specific domain knowledge, which is necessary prerequisite for human demonstrations, and can be designed using only sparse rewards.

The performance of the proposed SLD framework has been thoroughly examined on 13 robot manipulation environments of increasing complexity. These studies demonstrate that faster learning and higher success rate can be achieved through SLDs compared to existing methods. In our experiments, SLDs have been able to solve complex tasks, such as stacking 3 objects and manipulating non-rigid object with 5 tuples, whereas competing methods have failed. Remarkably, we have been able to outperform runner-up methods by a significant margin for complex Shadow’s hand manipulation tasks. Our approach leverages the availability of realistic 3D physics engines to

learn object-level locomotion policies. It is important to notice that this requirement does not restrict the applicability of the proposed approach on real world problems. The learnt manipulation policies do not require the locomotion policy to execute, hence they can be used when performing a manipulation task using real robots.

There are several aspects of the the proposed SLD approach that will be further investigated in follow-up work. We have noticed that when the interaction between the manipulating robot and the objects is very complex, the manipulation policy may be difficult to learn despite the fact that the locomotion policy is learnt successfully. For instance, in the case of the 5-tuple task with Fetch arm, the locomotion policy achieves a 100% success rate (as shown in Fig. 6(a)), whereas the manipulation policy does not always accomplish the task (as shown in Fig. 4(e) and Fig. 5). In such cases, since the ideal object locomotion depends heavily on the robot, leveraging the SLDs has a reduced benefit. Another limitation is that we assume $P(s_{obj}|\mu) \stackrel{d}{=} P(s_{obj}|\mu_{obj})$ (Assumption 2 in Section III-B), which may not hold for some tasks. For example, for pen manipulation tasks with Shadow’s hand, although the pen can rotate and translate itself to complete locomotion tasks (as shown in Fig. 6(b)), it is difficult for the robot to reproduce the same locomotion without dropping the pen. This issue can degrade the performance of the manipulation policy despite having obtained an optimal locomotion policy (see Fig. 4(b), Fig. 4(c) and Fig. 5). A possible solution would be to train the manipulation policy and locomotion policy jointly, and check whether the robot can reproduce the object locomotion suggested by the locomotion policy; a notion of “reachability” of object locomotion could be used to regularise the locomotion policy to enforce $P(s_{obj}|\mu) \stackrel{d}{=} P(s_{obj}|\mu_{obj})$; this direction will be explored in follow-up work.

In this paper we have adopted DDPG as the main training algorithm due to its widely reported effectiveness in robot control tasks. However other approaches would also be suitable for continuous action domain such as trust region policy optimisation (TRPO) [53], proximal policy optimisation (PPO) [54] and soft actor-critic [55]; their potential benefits over DDPG will be investigated, as well as alternative algorithms for learning the locomotion policy, including model-based methods [16], [56].

REFERENCES

- [1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [3] H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar, “Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost,” *arXiv preprint arXiv:1810.06045*, 2018.

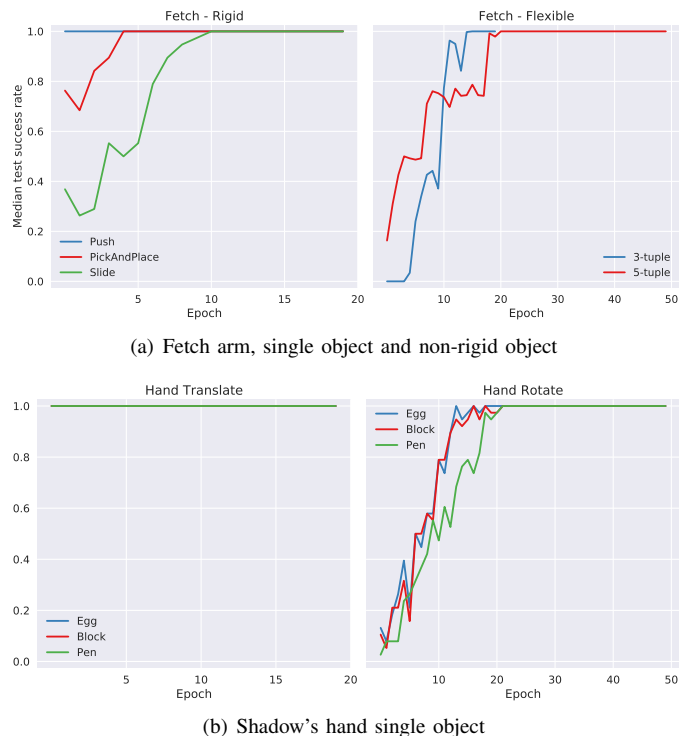


Fig. 6. Learning curves of locomotion policies. Recall that we train individual locomotion policies for translation and rotation in Shadow’s hand environments (see Section IV). We do not include Fetch arm multi object stacking environments as those environments reuse the policies learnt in Fetch arm single object environments (see Section V-D).

- [4] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, “Learning dexterous in-hand manipulation,” *arXiv preprint arXiv:1808.00177*, 2018.
- [5] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.
- [6] I. Popov, N. Heess, T. P. Lillicrap, R. Hafner, G. Barth-Maron, M. Vecerik, T. Lampe, Y. Tassa, T. Erez, and M. A. Riedmiller, “Data-efficient deep reinforcement learning for dexterous manipulation,” *CoRR*, vol. abs/1704.03073, 2017.
- [7] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, “Overcoming exploration in reinforcement learning with demonstrations,” in *International Conference on Robotics and Automation*, 2018, pp. 6292–6299.
- [8] S. Gu, E. Holly, T. P. Lillicrap, and S. Levine, “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates,” in *International Conference on Robotics and Automation*, 2017, pp. 3389–3396.
- [9] M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5055–5065.
- [10] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, “Exploration by random network distillation,” *arXiv preprint arXiv:1810.12894*, 2018.
- [11] M. A. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degraeve, T. V. de Wiele, V. Mnih, N. Heess, and J. T. Springenberg, “Learning by playing solving sparse reward tasks from scratch,” in *International Conference on Machine Learning*, 2018, pp. 4341–4350.
- [12] R. Houthoofd, X. Chen, Y. Duan, J. Schulman, F. D. Turck, and P. Abbeel, “VIME: variational information maximizing exploration,” in *Advances in Neural Information Processing Systems*, 2016.
- [13] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, “Large-scale study of curiosity-driven learning,” *arXiv preprint arXiv:1808.04355*, 2018.
- [14] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, “Curiosity-driven

- exploration by self-supervised prediction,” in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2017, pp. 16–17.
- [15] N. Savinov, A. Raichuk, R. Marinier, D. Vincent, M. Pollefeys, T. Lillicrap, and S. Gelly, “Episodic curiosity through reachability,” *arXiv preprint arXiv:1810.02274*, 2018.
 - [16] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine, “SOLAR: deep structured representations for model-based reinforcement learning,” in *International Conference on Machine Learning*, 2019.
 - [17] S. Ross, G. J. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 627–635.
 - [18] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, and J. Zhao, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.
 - [19] H. Xu, Y. Gao, F. Yu, and T. Darrell, “End-to-end learning of driving models from large-scale video datasets,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
 - [20] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo, and A. Gruslys, “Deep q-learning from demonstrations,” in *AAAI Conference on Artificial Intelligence*, 2018, pp. 3223–3230.
 - [21] A. Y. Ng and S. J. Russell, “Algorithms for inverse reinforcement learning,” in *International Conference on Machine Learning*, 2000, pp. 663–670.
 - [22] P. Abbeel and A. Y. Ng, “Apprenticeship learning via inverse reinforcement learning,” in *International Conference on Machine Learning*, 2004.
 - [23] C. Finn, S. Levine, and P. Abbeel, “Guided cost learning: Deep inverse optimal control via policy optimization,” in *International Conference on Machine Learning*, 2016, pp. 49–58.
 - [24] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems*, 2016, pp. 4565–4573.
 - [25] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller, “Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards,” *CoRR*, vol. abs/1707.08817, 2017.
 - [26] F. Sasaki, T. Yohira, and A. Kawaguchi, “Sample efficient imitation learning for continuous control,” in *International Conference on Learning Representations*, 2019.
 - [27] S. Reddy, A. D. Dragan, and S. Levine, “SQL: Imitation learning via regularized behavioral cloning,” *arXiv preprint arXiv:1905.11108*, 2019.
 - [28] D. H. Grollman and A. Billard, “Donut as i do: Learning from failed demonstrations,” in *IEEE International Conference on Robotics and Automation*, 2011.
 - [29] J. Zheng, S. Liu, and L. M. Ni, “Robust bayesian inverse reinforcement learning with sparse behavior noise,” in *AAAI Conference on Artificial Intelligence*, 2014.
 - [30] K. Shiarlis, J. Messias, and S. Whiteson, “Inverse reinforcement learning from failure,” in *International Conference on Autonomous Agents & Multiagent Systems*, 2016.
 - [31] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, “Reinforcement learning from imperfect demonstrations,” in *International Conference on Learning Representations*, 2018.
 - [32] D. S. Brown, W. Goo, P. Nagarajan, and S. Niekum, “Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations,” in *International Conference on Learning Representations*, 2019.
 - [33] S. Choi, K. Lee, and S. Oh, “Robust learning from demonstrations with mixed qualities using leveraged gaussian processes,” *IEEE Transactions on Robotics*, vol. 35, pp. 564–576, 2019.
 - [34] E. Todorov, T. Erez, and Y. Tassa, “Mujoco: A physics engine for model-based control,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
 - [35] N. Koenig and A. Howard, “Design and use paradigms for gazebo, an open-source multi-robot simulator,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.
 - [36] E. Coumans and Y. Bai, “Pybullet, a python module for physics simulation in robotics, games and machine learning,” <https://pybullet.org>, 2017.
 - [37] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *CoRR*, vol. abs/1509.02971, 2015.
 - [38] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
 - [39] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, “Deterministic policy gradient algorithms,” in *International Conference on Machine Learning*, 2014, pp. 387–395.
 - [40] C. J. C. H. Watkins and P. Dayan, “Technical note q-learning,” *Machine Learning*, vol. 8, pp. 279–292, 1992.
 - [41] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018.
 - [42] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba, “Multi-goal reinforcement learning: Challenging robotics environments and request for research,” *CoRR*, vol. abs/1802.09464, 2018.
 - [43] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: <http://arxiv.org/abs/1412.6980>
 - [44] D. Pomerleau, “ALVINN: an autonomous land vehicle in a neural network,” in *Advances in Neural Information Processing Systems*, 1988, pp. 305–313.
 - [45] Y. Duan, M. Andrychowicz, B. C. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, “One-shot imitation learning,” in *Advances in Neural Information Processing Systems*, 2017, pp. 1087–1098.
 - [46] N. D. Ratliff, J. A. Bagnell, and S. S. Srinivasa, “Imitation learning for locomotion and manipulation,” in *International Conference on Humanoid Robots*, 2007, pp. 392–397.
 - [47] Z. Wang, J. Merel, S. Reed, G. Wayne, N. de Freitas, and N. Heess, “Robust imitation of diverse behaviors,” *arXiv preprint arXiv:1707.02747*, 2017.
 - [48] T. Schaul, D. Horgan, K. Gregor, and D. Silver, “Universal value function approximators,” in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 1312–1320.
 - [49] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus, “Intrinsic motivation and automatic curricula via asymmetric self-play,” in *International Conference on Learning Representations*, 2018.
 - [50] C. Florensa, D. Held, X. Geng, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” in *International Conference on Machine Learning*, 2018, pp. 1514–1523.
 - [51] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, “Visual reinforcement learning with imagined goals,” in *Advances in Neural Information Processing Systems*, 2018, pp. 9209–9220.
 - [52] J. Schmidhuber, “Curious model-building control systems,” in *IEEE International Joint Conference on Neural Networks*. IEEE, 1991, pp. 1458–1463.
 - [53] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in *International Conference on Machine Learning*, 2015, pp. 1889–1897.
 - [54] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
 - [55] 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*, 2018.
 - [56] K. Chua, R. Calandra, R. McAllister, and S. Levine, “Deep reinforcement learning in a handful of trials using probabilistic dynamics models,” in *International Conference on Machine Learning*, 2019.