# Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments

Vinicius G. Goecks
Texas A&M University
US Army Research Laboratory
College Station, Texas
vinicius.goecks@tamu.edu

Gregory M. Gremillion
US Army Research Laboratory
Adelphi, Maryland
gregory.m.gremillion.civ@mail.mil

Vernon J. Lawhern
US Army Research Laboratory
Aberdeen, Maryland
vernon.j.lawhern.civ@mail.mil

John Valasek
Texas A&M University
College Station, Texas
valasek@tamu.edu

Nicholas R. Waytowich
US Army Research Laboratory
Columbia University
Aberdeen, Maryland
nicholas.r.waytowich.civ@mail.mil

## ABSTRACT

This paper investigates how to efficiently transition and update policies, trained initially with demonstrations, using off-policy actor-critic reinforcement learning. It is well-known that techniques based on Learning from Demonstrations, for example behavior cloning, can lead to proficient policies given limited data. However, it is currently unclear how to efficiently update that policy using reinforcement learning as these approaches are inherently optimizing different objective functions. Previous works have used loss functions, which combine behavior cloning losses with reinforcement learning losses to enable this update. However, the components of these loss functions are often set anecdotally, and their individual contributions are not well understood. In this work, we propose the Cycle-of-Learning (CoL) framework that uses an actor-critic architecture with a loss function that combines behavior cloning and 1-step Q-learning losses with an off-policy pre-training step from human demonstrations. This enables transition from behavior cloning to reinforcement learning without performance degradation and improves reinforcement learning in terms of overall performance and training time. Additionally, we carefully study the composition of these combined losses and their impact on overall policy learning. We show that our approach outperforms state-of-the-art techniques for combining behavior cloning and reinforcement learning for both dense and sparse reward scenarios. Our results also suggest that directly including the behavior cloning loss on demonstration data helps to ensure stable learning and ground future policy updates.

## CCS CONCEPTS

• **Computing methodologies** → **Learning from demonstrations**; **Reinforcement learning**; *Artificial intelligence*; • **Human-centered computing** → *Human computer interaction (HCI)*; *Interaction design process and methods*;

## KEYWORDS

Human-robot/agent interaction; Agent-based analysis of human interaction; Machine learning for robotics; Reinforcement Learning

**ACM Reference Format:**
Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich. 2020. Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments. In *Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), Auckland, New Zealand, May 9–13, 2020,* IFAAMAS, 9 pages.

## 1 INTRODUCTION

Reinforcement Learning (RL) has yielded many recent successes in solving complex tasks that meet and exceed the capabilities of human counterparts, demonstrated in video game environments [19], robotic manipulators [1], and various open-source simulated scenarios [17]. However, these RL approaches are sample inefficient and slow to converge to this impressive behavior, limited significantly by the need to explore potential strategies through trial and error, which produces initial performance significantly worse than human counterparts. The resultant behavior that is initially random and slow to reach proficiency is poorly suited for real-world applications such as physically embodied ground and air vehicles, or in scenarios where sufficient capability must be achieved in short time spans. In such situations, the random exploration of the state space of an untrained agent can result in unsafe behaviors and catastrophic failure of a physical system, potentially resulting in unacceptable damage or downtime. Similarly, slow convergence of the agent's performance requires exceedingly many interactions with the environment, which is often prohibitively difficult or infeasible for physical systems that are subject to energy constraints, component failures, and operation in dynamic or adverse environments. These sample efficiency pitfalls of RL are exacerbated even further when trying to learn in the presence of sparse rewards, often leading to cases where RL can fail to learn entirely.

One approach for overcoming these limitations is to utilize demonstrations of desired behavior from a human data source (or potentially some other agent) to initialize the learning agent

*Proc. of the 19th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2020), B. An, N. Yorke-Smith, A. El Fallah Seghrouchni, G. Sukthankar (eds.), May 9–13, 2020, Auckland, New Zealand.* 

to a significantly higher level of performance than is yielded by a randomly initialized agent. This is often termed Learning from Demonstrations (LfD) [2], which is a subset of imitation learning that seeks to train a policy to imitate the desired behavior of another policy or agent. LfD leverages data (in the form of state-action tuples) collected from a demonstrator for supervised learning, and can be used to produce an agent with qualitatively similar behavior in a relatively short training time and with limited data. This type of LfD, called Behavior Cloning (BC), attempts to learn a mapping between the state-action pairs contained in the set of demonstrations to mimic the behavior of the demonstrator. LfD also encompasses other learning modalities such as inverse reinforcement learning (IRL), where it is desired to learn the reward function that the agent, or demonstrator, is optimizing to perform the task [25].

Though BC techniques do allow for the relatively rapid learning of behaviors that are comparable to that of the demonstrator, they are limited by the quality and quantity of the demonstrations provided and are only improved by providing additional, high-quality demonstrations. In addition, BC is plagued by the distributional drift problem in which a mismatch between the learned policy distribution of states and the distribution of states in the training set can cause errors that propagate over time and lead to catastrophic failures. By combining BC with subsequent RL, it is possible to address the drawbacks of either approach, initializing a significantly more capable and safer agent than with random initialization, while also allowing for further self-improvement without needing to collect additional data from a human demonstrator. However, it is currently unclear how to effectively update a policy initially trained with BC using RL as these approaches are inherently optimizing different objective functions. Previous works have used loss functions that combine BC losses with RL losses to enable this update, however, the components of these loss functions are often set anecdotally and their individual contributions are not well understood.

In this work, we propose the Cycle-of-Learning (CoL) framework, which uses an actor-critic architecture with a loss function that combines behavior cloning and 1-step Q-learning losses with an off-policy algorithm, and a pre-training step to learn from human demonstrations. Unlike previous approaches to combine BC with RL, such as Rajeswaran et al. [24], our approach uses an actor-critic architecture to learn both a policy and value function from the human demonstration data, which we show, speeds up learning. Additionally, we perform a detailed component analysis of our method to investigate the individual contributions of pre-training, combined losses, and sampling methods of the demonstration data and their effects on transferring from BC to RL. To summarize, the main contribution of this work are:

- We introduce an actor-critic based method, that combines pre-training as well as combined loss functions to learn both a policy and value function from demonstrations, to enable transition from behavior cloning to reinforcement learning.
- We show that our method can transfer from BC to RL without performance degradation while improving upon existing state-of-the-art BC to RL algorithms in terms of overall performance and training time.
- We perform a detailed analysis to investigate the contributions of the individual components in our method.

Our results show that our approach outperforms BC, Deep Deterministic Policy Gradients (DDPG), and Demonstration Augmented Policy Gradient (DAPG) in two different application domains for both dense- and sparse-reward settings. Our results also suggest that directly including the behavior cloning loss on demonstration data helps to ensure stable learning and ground future policy updates, and that a pre-training step enables the policy to start at a performance level greater than behavior cloning.

## 2 PRELIMINARIES

We adopt the standard Markov Decision Process (MDP) formulation for sequential decision making [31], which is defined as a tuple $(S, A, R, P, \gamma)$, where $S$ is the set of states, $A$ is the set of actions, $R(s, a)$ is the reward function, $P(s'|s, a)$ is the transition probability function and $\gamma$ is a discount factor. At each state $s \in S$, the agent takes an action $a \in A$, receives a reward $R(s, a)$ and arrives at state $s'$ as determined by $P(s'|s, a)$. The goal is to learn a behavior policy $\pi$ which maximizes the expected discounted total reward. This is formalized by the Q-function, sometimes referred to as the state-action value function:

$$Q^{\pi}(s, a) = \mathbb{E}_{a_t \sim \pi}\left[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)\right]$$

taking the expectation over trajectories obtained by executing the policy $\pi$ starting at $s_0 = s$ and $a_0 = a$.

Here we focus on actor-critic methods which seek to maximize

$$J(\theta) = \mathbb{E}_{s \sim \mu}[Q^{\pi(.|\theta)}(s, \pi(s|\theta))]$$

with respect to parameters $\theta$ and an initial state distribution $\mu$. The Deep Deterministic Policy Gradient (DDPG) [17] is an off-policy actor-critic reinforcement learning algorithm for continuous action spaces, which calculates the gradient of the Q-function with respect to the action to train the policy. DDPG makes use of a replay buffer to store past state-action transitions and target networks to stabilize Q-learning [19]. Since DDPG is an off-policy algorithm, it allows for the use of arbitrary data, such as demonstrations from another source, to update the policy. A demonstration trajectory is a tuple $(s, a, r, s')$ of state $s$, action $a$, the reward $r = R(s, a)$ and the transition state $s'$ collected from a demonstrator's policy. In most cases these demonstrations are from a human observer, although in principle these demonstrations can come from any existing agent or policy.

## 3 RELATED WORK

Several works have shown the efficacy of combining behavior cloning with reinforcement learning across a variety of tasks. One of the earliest works in this area was by Schaal [27], who studied demonstration learning and model-based reinforcement learning and their application to classical tasks such as cart-pole. Similarly, Atkeson and Schaal [3], in a robotic arm swinging up a pendulum task, trained a dynamics model and reward function from human demonstrations to learn a policy and improve it with reinforcement learning. Kim et al. [14] used expert samples to constrain the approximate policy iteration step and learn a value function, parametrized by linear radial basis functions (RBF), with convex optimization. Recent work by Hester et al. [9], known as Deep Q-learning from

Demonstrations (DQfD), combined behavior cloning with deep Q-learning [19] to learn policies for Atari games by leveraging a loss function that combines a large-margin supervised learning loss function, 1-step Q-learning loss, and an $n$-step Q-learning loss function that helps ensure the network satisfies the Bellman equation. This work was extended to continuous action spaces by Večerík et al. [33] with DDPG from Demonstrations (DDPGfD), who proposed an extension of DDPG [17] that uses human demonstrations, and applied their approach to object manipulation tasks for both simulated and real robotic environments. The loss functions for these methods include the $n$-step Q-learning loss, which is known to require on-policy data to accurately estimate. Similar work by Nair et al. [21] combined behavior cloning-based demonstration learning, goal-based reinforcement learning, and DDPG for robotic manipulation of objects in a simulated environment.

The Normalized Actor Critic [6] uses principles from maximum entropy reinforcement learning [8] and proposes a learning objective which better normalizes the Q-function learned from demonstration data. In addition they proposed a single unified loss function as opposed to a combined loss function of supervised and reinforcement losses and showed superior performance versus existing works in a Minecraft task and two 3D driving tasks. Policy Optimization with Demonstrations (POfD) [13] specifies a demonstration learning approach using an adversarial learning objective, seeking to minimize the difference between the learned policy and the demonstration policy when the reward signal is sparse, an approach similar in nature to Generative Adversarial Imitation Learning (GAIL) [11].

A method that is very similar to ours is the Demonstration Augmented Policy Gradient (DAPG) [24], a policy-gradient method that uses behavior cloning as a pre-training step together with an augmented loss function with a heuristic weight function that interpolates between the policy gradient loss, computed using the Natural Policy Gradient [12], and behavior cloning loss. They apply their approach across four different robotic manipulations tasks using a 24 Degree-of-Freedom (DoF) robotic hand in a simulator and show that DAPG outperforms DDPGfD [33] across all tasks. Their work also showed that behavior cloning combined with Natural Policy Gradient performed very similarly to DAPG for three of the four tasks considered, showcasing the importance of using a behavior cloning loss both in pre-training and policy training.

In summary, when compared to the main related literature, the Cycle-of-Learning (CoL) algorithm differs from existing algorithms in several ways. First, CoL uses an actor-critic architecture, as opposed to the policy gradient algorithm proposed by DAPG [24]. The actor-critic architecture allows the integration of additional human interaction modalities during training as, for example, evaluative feedback to update the critic and human interventions to update the actor; Second, CoL introduces a pre-training phase the combined loss function and the expert demonstrations are used to train the actor and critic network before interacting with the environment, which is not present on state-of-the-art works such as Nair et al. [21] and Pohlen et al. [22]. Third, CoL learns in continuous action-space environments as opposed to discrete action-spaces as was done in Hester et al. [9], Večerík et al. [33], and Pohlen et al. [22].

# 4 PROPOSED APPROACH

The Cycle-of-Learning (CoL) framework is a method for leveraging multiple modalities of human input to improve the training of RL agents. These modalities can include human demonstrations, i.e. human-provided exemplar behaviors, human interventions, i.e. interdictions in agent behavior with subsequent partial demonstrations, and human evaluations, i.e. sparse indications of the quality of agent behavior. These individual mechanisms of human interaction have been previously shown to provide various benefits in learning performance and efficiency [7, 16, 18, 26, 34]. The successful integration of these disparate techniques, which would leverage their complementary characteristics, requires a learning architecture that allows for optimization of common objective functions and consistent representations. An actor-critic framework with a combined loss function, as presented in this work, is such an architecture.

In this paper, we focus on extending the Cycle-of-Learning framework to tackle the known issue of transitioning BC policies to RL by utilizing an actor-critic architecture with a combined BC+RL loss function and pre-training phase for continuous state-action spaces, that can learn in both dense- and sparse-reward environments. The main advantage of our method is the use of an off-policy, actor-critic architecture to pre-train both a policy and value function, as well as continued re-use of demonstration data during agent training, which reduces the amount of interactions needed between the agent and environment. This is an important aspect especially for robotic applications or real-world systems where interactions can be costly.

The combined loss function consists of the following components: an expert behavior cloning loss that drives the actor's actions toward previous human trajectories, 1-step return Q-learning loss to propagate values of human trajectories to previous states, the actor loss, and a $L_2$ regularization loss on the actor and critic to stabilize performance and prevent over-fitting during training. The implementation of each loss component and their combination are defined as follows:

- **Expert behavior cloning loss ($\mathcal{L}_{BC}$):** Given expert demonstration subset $\mathcal{D}_E$ of continuous states and actions $s^E$ and $a^E$ visited by the expert during a task demonstration over $T$ time steps

$$\mathcal{D}_E = \left\{ s_0^E, a_0^E, s_1^E, a_1^E, ..., s_T^E, a_T^E \right\}, \tag{1}$$

  a behavior cloning loss (mean squared error) from demonstration data $\mathcal{L}_{BC}$ can be written as

$$\mathcal{L}_{BC}(\theta_\pi) = \frac{1}{2} \left( \pi(s_t|\theta_\pi) - a_t^E \right)^2 \tag{2}$$

  in order to minimize the difference between the actions predicted by the actor network $\pi(s_t)$, parametrized by $\theta_\pi$, and the expert actions $a_{E_t}$ for a given state vector $s_t$.
- **1-step return Q-learning loss ($\mathcal{L}_1$):** The 1-step return $R_1$ can be written in terms of the critic network $Q$, parametrized by $\theta_Q$, as

$$R_1 = r_t + \gamma Q(s_{t+1}, \pi(s_{t+1}|\theta_\pi)|\theta_Q). \tag{3}$$

  In order to satisfy the Bellman equation, we minimize the difference between the predicted Q-value and the observed

return from the 1-step roll-out for a batch of sampled states **s**:

$$\mathcal{L}_{Q_1}(\theta_Q) = \frac{1}{2}\left(R_1 - Q(\mathbf{s}, \pi(\mathbf{s}|\theta_\pi)|\theta_Q)\right)^2. \quad (4)$$

- **Actor Q-loss ($\mathcal{L}_A$):** It is assumed that the critic function $Q$ is differentiable with respect to the action. Since we want to maximize the Q-values for the current state, the actor loss became the negative of the Q-values predicted by the critic for a batch of sampled states **s**:

$$\mathcal{L}_A(\theta_\pi) = -Q(\mathbf{s}, \pi(\mathbf{s}|\theta_\pi)|\theta_Q). \quad (5)$$

- *L2* **regularization ($\mathcal{L}_{L2}$):** We also add a L2 regularization term for the actor and critic weights to prevent overfitting and control model complexity:

$$\mathcal{L}_{L2}(\theta_\pi) = \theta_\pi^T \theta_\pi, \quad (6)$$

$$\mathcal{L}_{L2}(\theta_Q) = \theta_Q^T \theta_Q. \quad (7)$$

Combining the above loss functions for the Cycle-of-Learning becomes

$$\mathcal{L}_{CoL}(\theta_Q, \theta_\pi) = \lambda_{BC}\mathcal{L}_{BC}(\theta_\pi) + \lambda_A \mathcal{L}_A(\theta_\pi) + \lambda_{Q_1}\mathcal{L}_{Q_1}(\theta_Q) + \lambda_{L2Q}\mathcal{L}_{L2}(\theta_Q) + \lambda_{L2\pi}\mathcal{L}_{L2}(\theta_\pi). \quad (8)$$

Our approach starts by collecting contiguous trajectories from expert policies and stores the current and subsequent state-actions pairs, reward received, and task completion signal in a permanent expert memory buffer $\mathcal{D}_E$. During the pre-training phase, the agent samples a batch of trajectories from the expert memory buffer $\mathcal{D}_E$ containing expert trajectories to perform updates on the actor and critic networks using the same combined loss function (Equations 8). This procedure shapes the actor and critic initial distributions to be closer to the expert trajectories and eases the transition from policies learned through expert demonstration to reinforcement learning.

After the pre-training phase, the policy is allowed to roll-out and collect its first on-policy samples, which are stored in a separate first-in-first-out memory buffer with only the agent's samples. After collecting a given number of on-policy samples, the agent samples a batch of trajectories comprising 25% of samples from the expert memory buffer and 75% from the agent's memory buffer. This fixed ratio guarantees that each gradient update is grounded by expert trajectories. We opted to use a fixed buffer ratio as in the Ape-X DQfD [22], one of the extensions of DQfD [9], which is claimed to be the first RL algorithm to solve the first level of Montezuma Revenge, a challenging ATARI task with sparse rewards. In our experiments, showed in Section 5, we also compared this fixed buffer ratio with the traditional Prioritized Experience Replay (PER) method and showed that the fixed buffer ratio outperforms PER in the sparse reward scenario. If a human demonstrator is used, they can intervene at any time the agent is executing their policy, and add this new trajectories to the expert memory buffer. Samples collected by the agent are added to the agent memory buffer, as usual.

After sampling a batch of trajectories from the expert and agent buffers, we perform model updates using the CoL combined loss. This process is repeated after each interaction with the environment.

**Algorithm 1** Cycle-of-Learning (CoL): Transitioning from Demonstrations to Reinforcement Learning

Input:
  Environment *env*, training steps $T$, data collection steps $M$, batch size $N$, pre-training steps $L$, CoL hyperparameters $\lambda_{Q_1}, \lambda_{BC}, \lambda_A, \lambda_{L2Q}, \lambda_{L2\pi}, \tau$, and expert trajectories $\mathcal{D}_E$ (if available).
Output:
  Trained actor $\pi(s|\theta_\pi)$ and critic $Q(s, \pi|\theta_Q)$ networks.
Randomly initialize:
  Actor network $\pi(s|\theta_\pi)$ and its target $\pi'(s|\theta_{\pi'})$ weights.
  Critic network $Q(s, \pi|\theta_Q)$ and its target $Q'(s, \pi'|\theta_{Q'})$ weights.
Initialize empty agent and expert replay buffers $\mathcal{R}$ and $\mathcal{R}_E$.
Load $\mathcal{R}$ and $\mathcal{R}_E$ with expert trajectories $\mathcal{D}_E$, if available.
**for** pre-training steps = 1, ..., $L$ **do**
  Call *TrainUpdate*() procedure.
**for** training steps = 1, ..., $T$ **do**
  Reset *env* and receive initial state $s_0$.
  **for** data collection steps = 1, ..., $M$ **do**
    Select action $a_t = \pi(s_t|\theta_\pi)$ according to policy.
    Perform action $a_t$ and observe reward $r_t$ and next state $s_{t+1}$.
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{R}$.
    **if** End of episode **then**
      Reset *env* and receive initial state $s_0$.
  Call *TrainUpdate*() procedure.
**procedure** TrainUpdate()
  **if** Pre-training **then**
    Randomly sample $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from the expert replay buffer $\mathcal{R}_E$.
  **else**
    Randomly sample $N * 0.25$ transitions $(s_i, a_i, r_i, s_{i+1})$ from the expert replay buffer $\mathcal{R}_E$ and $N * 0.75$ transitions from the agent replay buffer $\mathcal{R}$.
  Compute $\mathcal{L}_{Q_1}(\theta_Q)$, $\mathcal{L}_{BC}(\theta_\pi)$, $\mathcal{L}_A(\theta_\pi)$, $\mathcal{L}_{L2}(\theta_Q)$, $\mathcal{L}_{L2}(\theta_\pi)$
  Update actor and critic networks according to Equation 8.
  Update target networks:

$$\theta_{\pi'} \leftarrow \tau\theta_\pi + (1-\tau)\theta_{\pi'},$$
$$\theta_{Q'} \leftarrow \tau\theta_Q + (1-\tau)\theta_{Q'}.$$

The proposed method is summarized in the pseudocode shown in Algorithm 1.

## 5 EXPERIMENTAL SETUP AND RESULTS

### 5.1 Experimental Setup

As described in the previous sections, in our approach, the Cycle-of-Learning (CoL), we collect contiguous trajectories from expert policies and store them in a permanent memory buffer. The policy is allowed to roll-out and is trained with a combined loss from a mix of demonstration and agent data, stored in a separate first-in-first-out buffer. We validate our approach in three environments with continuous observation- and action-space: LunarLanderContinuous-v2

[4] (dense and sparse reward cases) and a custom quadrotor landing task [7] implemented using Microsoft AirSim [29].

The dense reward case of LunarLanderContinuous-v2 is the standard environment provided by OpenAI Gym library [4]: the state space consists of a eight-dimensional continuous vector with inertial states of the lander, the action space consists of a two-dimensional continuous vector controlling main and side thrusters, and the reward is given at every step based on the relative motion of the lander with respect to the landing pad (bonus reward is given when the landing is completed successfully). The sparse reward case is a custom modification with the same reward scheme and state-action space, however the reward is stored during the policy roll-out and is only given to the agent when the episode ends and is zero otherwise. The custom quadrotor landing task is a modified version of the environment proposed by Goecks et al. [7], implemented using Microsoft AirSim [29], which consists of landing a quadrotor on a static landing pad in a simulated gusty environment, as seen in Figure 1. The state space consists of a fifteen-dimensional continuous vector with inertial states of the quadrotor and visual features that represent the landing pad image-frame position and radius as seen by a downward-facing camera. The action space is a four-dimensional continuous vector that sends velocity commands for throttle, roll, pitch, and yaw. Wind is modeled as noise applied directly to the actions commanded by the agent and follows a temporal-based, instead of distance-based, discrete wind gust model [20] with 65% probability of encountering a wind gust at each time step. This was done to induce additional stochasticity in the environment. The gust duration is uniformly sampled to last between one to three real time seconds and can be imparted in any direction, with maximum velocity of half of what can be commanded by the agent along each axis. This task has a sparse-reward scheme (reward $R$ is given at the end of the episode, and is zero otherwise) based on the relative distance $r_{rel}$ between the quadrotor and the center of the landing pad at the final time step of the episode:

$$R = \frac{1}{1 + r_{rel}^2}.$$

Although the goals of the two tasks are similar, the environments are different in terms of state- and action-spaces and physics (2d vs 3d, with and without wind effects). Additionally, in these environments it was possible to collect human demonstrations of the task. We studied using standard benchmark environments as, for example, MuJoCo and PyBullet locomotion tasks, however, due to the nature of the tasks and number of controls, collecting human demonstrations were not feasible.

The hyperparameters used in CoL for each environment, and how to tune them properly, are described in the project page available online[1].

The baselines that we compare our approach to are Deep Deterministic Policy Gradient (DDPG) [17, 30], Demonstration Augmented Policy Gradient (DAPG) [24], and traditional behavior cloning (BC). For the DDPG baseline we used an open-source implementation by Stable Baselines [10]. The hyperparameters used concur with the original DDPG publication [17]: actor and critic networks with 2 hidden layers with 400 and 300 units respectively,

**Figure 1: Screenshot of AirSim environment and landing task. Inset image in lower right corner: downward-facing camera view used for extracting the position and radius of the landing pad, which is part of the state space.**

optimized using Adam [15] with learning rate of $10^{-4}$ for the actor and $10^{-3}$ for the critic, discount factor of $\gamma = 0.99$, trained with minibatch size of 64, and replay buffer size of $10^6$. Exploration noise was added to the action following an Ornstein-Uhlenbeck process [32] with mean of 0.15 and standard deviation of 0.2. For the DAPG baseline we used an official release of the DAPG codebase from the authors [2]. The policy is represented by a deep neural network with three hidden layers of 128 units each, pre-trained with behavior cloning for 100 epochs, with a batch size of 32 samples, and learning rate of $10^{-3}$, $\lambda_0 = 0.01$, and $\lambda_1 = 0.99$. The BC policies are trained by minimizing the mean squared error between the expert demonstrations and the output of the model. The policies consist of a fully-connected neural network with 3 hidden layers with 128 units each and exponential linear unit (ELU) activation function [5]. The BC policy was evaluated for 100 episodes which was used to calculate the mean and standard error of the performance of the policy.

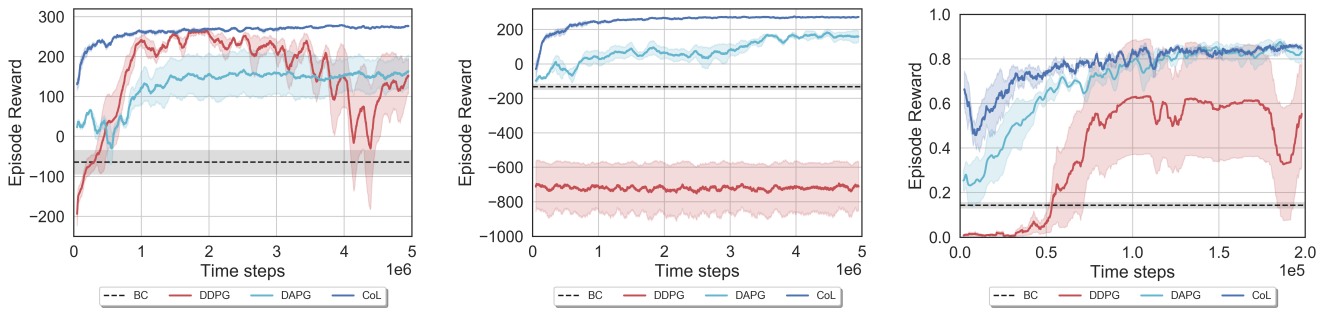
All baselines that rely on demonstrations, namely BC, DAPG, and CoL, use the same human trajectories collected in the LunarLanderContinuous-v2 and custom Microsoft AirSim environment.

## 5.2 Experimental Results

The comparative performances of the CoL against the baseline methods (BC, DDPG and DAPG) for the LunarLanderContinuous-v2 environment are presented via their training curves in Figure 2a, using the standard dense reward. The mean reward of the BC pre-trained from the human demonstrations is also shown for reference, and its standard error is shown by the shaded band. The CoL reward initializes to values at or above the BC and steadily improves throughout the reinforcement learning phase. Conversely, the DDPG RL baseline initially returns rewards lower than the BC and slowly improves until its performance reaches similar levels to the CoL after approximately one million steps. However, this baseline never performs as consistently as the CoL and eventually begins to diverge, losing much of its performance gains after about

[1]Cycle-of-Learning project page: https://vggoecks.com/cycle-of-learning/.

[2]DAPG implementation: https://github.com/aravindr93/hand_dapg [23].




(a) (b) (c)

**Figure 2: Comparison of CoL, BC, DDPG, and DAPG for 3 random seeds (bold line representing the mean and shaded area the standard error) in the (a) dense- and (b) sparse-reward LunarLanderContinuous-v2 environment, and the (c) sparse-reward Microsoft AirSim quadrotor landing environment.**

four million steps. The DAPG baseline initial performance, similar to the CoL, surpasses behavior cloning due to the pre-training phase and slowly converges to a high score, although slower than the CoL.

When using sparse rewards, meaning the rewards generated by the LunarLanderContinuous-v2 environment are provided only at the last time step of each episode, the performance improvement of the CoL relative to the DDPG and DAPG baselines is even greater (Figure 2b). The performance of the CoL is qualitatively similar during training to that of the dense case, with an initial reward roughly equal to or greater than that of the BC and a consistently increasing reward. Conversely, the performance of the DDPG baseline is greatly diminished for the sparse reward case, yielding effectively no improvement throughout the whole training period. The training of the DAPG does not deteriorate when compared to the dense reward case, however, the performance does not match CoL for the specified training time.

The results for the more realistic and challenging AirSim quadrotor landing environment (Figure 2c) illustrate a similar trend. The CoL initially returns rewards above the BC, DDPG, and DAPG baselines and steadily increases its performance, with DAPG converging at end to a similar level of performance. The DDPG baseline practically never succeeds and subsequently fails to learn a viable policy, while displaying greater variance in performance when compared to CoL and DAPG. Noting that successfully landing on the target would generate a sparse episode reward of approximately 0.64, it is clear that these baseline algorithms, with exception of DAPG, rarely generate a satisfactory trajectory for the duration of training.

## 5.3 Component Analysis

Several component analyses were performed to evaluate the impact of each of the critical elements of the CoL on learning. These respectively include the effects of pre-training, the combined loss function, and the sample composition of the experience replay buffer. The results of each analysis are shown in Figures 3-5 and are summarized in Table 1.

*5.3.1 Effects of Pre-Training.* To determine the effects of pre-training on performance we compare the standard CoL against an implementation without this pre-training phase, where the number of pre-training steps $L = 0$, denoted as *CoL-PT*. The complete combined loss, as seen in Equations 8 is used during the reinforcement learning phase. This condition assesses the impact on learning performance of not pre-training the agent, while still using the combined loss in the RL phase. As seen in Figure 3, this condition differs from the baseline CoL in its initial performance being worse, i.e. significantly below the BC, but does reach similar rewards after several hundred thousand steps, exhibiting the same consistent response during training thereafter. Effectively, this highlights that the benefit of pre-training is improved initial response and significant speed gain in reaching steady-state performance level, without qualitatively impacting the long-term training behavior.

*5.3.2 Effects of Combined Loss.* To determine the effects of the combined loss function on performance we compare the standard CoL against two alternate learning implementations: 1) the CoL without the behavioral cloning expert loss on the actor ($\lambda_{BC} := 0$) during both pre-training and RL phases, denoted as *CoL-BC*, and 2) standard BC followed by DDPG using standard loss functions, denoted as *BC+DDPG*. For the implementation of the CoL without the behavior cloning loss (*CoL-BC*), the critic loss remains the same as in Equation 8 for both training phases. This condition assesses the impact on learning performance of the behavior cloning loss component $\mathcal{L}_{BC}$, given otherwise consistent loss functions in both pre-training and RL phases. As seen in Figure 4, this condition (purple, dashed) improves upon the CoL-PT condition (Figure 3) in its initial reward return and similarly achieves comparable performance to the baseline CoL in the first few hundred thousand steps, but then steadily deteriorates as training continues, with several catastrophic losses in performance. This result makes clear that the behavioral cloning loss is an essential component of the combined loss function toward maintaining performance throughout training, anchoring the learning to some previously demonstrated behaviors that are sufficiently proficient.

**Table 1: Method Comparison on LunarLanderContinuous-v2 environment, dense-reward case**

| Method | Pre-Training Loss | Training Loss | Buffer Type | Average Reward |
|---|---|---|---|---|
| CoL | $\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$ | $\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$ | Fixed Ratio | 261.80 ± 22.53 |
| CoL-PT | None | $\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$ | Fixed Ratio | 253.24 ± 46.50 |
| CoL+PER | $\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$ | $\mathcal{L}_{Q_1} + \mathcal{L}_A + \mathcal{L}_{BC}$ | PER | 245.24 ± 37.66 |
| DAPG | $\mathcal{L}_{BC}$ | Augmented Policy Gradient | None | 127.99 ± 37.28 |
| DDPG | None | $\mathcal{L}_{Q_1} + \mathcal{L}_A$ | Uniform | 152.98 ± 69.45 |
| BC | $\mathcal{L}_{BC}$ | None | None | -48.83 ± 27.68* |
| BC+DDPG | $\mathcal{L}_{BC}$ | $\mathcal{L}_{Q_1} + \mathcal{L}_A$ | Uniform | -57.38 ± 50.11 |
| CoL-BC | $\mathcal{L}_{Q1} + \mathcal{L}_A$ | $\mathcal{L}_{Q_1} + \mathcal{L}_A$ | Fixed Ratio | -105.65 ± 196.85 |

Summary of learning methods. Enumerated for each method are all non-zero loss components (excluding regularization), buffer type, and average and standard error of the reward throughout training (after pre-training) across the three seeds, evaluated with dense reward in LunarLanderContinuous-v2 environment. *For BC, these values are computed from 100 evaluation trajectories of the final pre-trained agent.

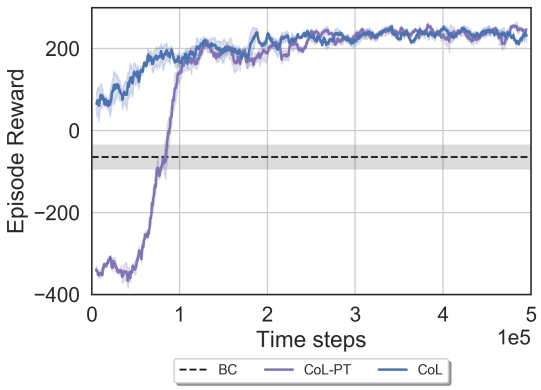



**Figure 3: Effects of the pre-training phase in the Cycle-of-Learning. Results for 3 random seeds (bold line representing the mean and shaded area the standard error) showing component analysis in LunarLanderContinuous-v2 environment comparing pre-trained Cycle-of-Learning (CoL curve) against the Cycle-of-Learning without the pre-training phase (CoL-PT curve) and the behavior cloning (BC) baseline.**

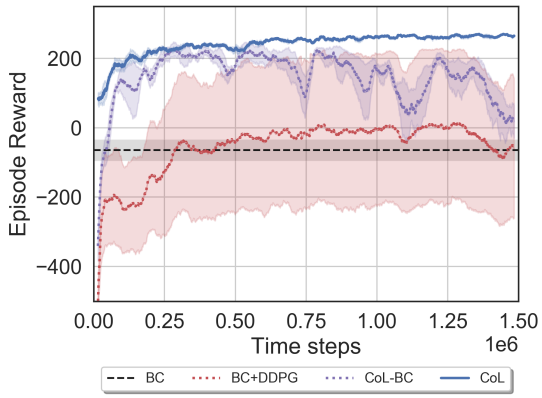



**Figure 4: Effects of the combined loss in the Cycle-of-Learning. Results for 3 random seeds (bold line representing the mean and shaded area the standard error) showing component analysis in LunarLanderContinuous-v2 environment comparing complete Cycle-of-Learning (CoL), CoL without the expert behavior cloning loss (CoL-BC), and pre-training with BC followed by DDPG without combined loss (BC+DDPG).**

The second of these comparative implementations that illustrate the effects of the combined loss is the behavior cloning with subsequent DDPG (*BC+DDPG*) condition, which utilized standard loss functions (Equations 2, 4, and 5) rather than the CoL combined loss in both phases (Equation 8). Pre-training of the actor with BC uses only the regression loss, as seen in Equation 2. DDPG utilizes standard loss functions for the actor and critic, as seen in Lillicrap et al. [17]. The BC+DDPG condition assesses the impact on learning performance of standardized loss functions rather than our combined loss functions across both training phases. The BC+DDPG condition (Figure 4; red, dashed) produces initial rewards below the BC response and subsequently improves in performance only to an average level similar to that of the BC and is much less stable in its response throughout training, as indicated by the wide standard error band. This result indicates that simply sequencing standard BC and RL algorithms results in significantly worse performance and stability even after millions of training steps, emphasizing the value of a consistent combined loss function across all training phases.

*5.3.3 Effects of Human Experience Replay Sampling.* To determine the effects of the different sampling techniques of the experience replay buffer on performance we compare the standard CoL, which utilizes a fixed ratio buffer of samples comprising 25% expert data and 75% agent data, against an implementation with

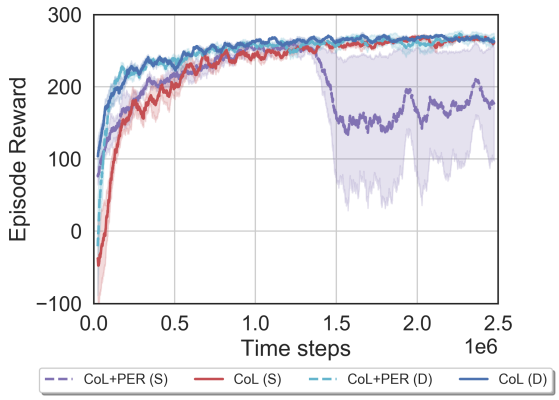



**Figure 5: Effects of human experience replay sampling in the Cycle-of-Learning. Results for 3 random seeds (bold line representing the mean and shaded area the standard error) showing ablation study in LunarLanderContinuous-v2 environment, dense (D) and sparse (S) reward cases, comparing complete Cycle-of-Learning (CoL) trained with fixed ratio of expert and agent samples and complete Cycle-of-Learning using Prioritized Experience Replay (CoL+PER) with a variable ratio of expert and agent samples ranked based on their temporal difference error.**

Prioritized Experience Replay (PER) [28], with a data buffer prioritized by the magnitude of each transition's temporal difference (TD) error, denoted as *CoL+PER*. The comparative performance of these implementations, for both the dense- (D) and sparse-reward (S) cases of the LunarLanderContinuous-v2 scenario, are shown in Figure 5. For the dense-reward condition, there is no significant difference in the learning performance between the fixed ratio and PER buffers. However, for the sparse-reward case of the *CoL+PER* implementation, the learning breaks down after approximately 1.3 million training steps, resulting in a significantly decreased performance thereafter. This result illustrates that the fixed sampling ratio for the replay buffer in the standard CoL is a more robust mechanism of incorporating experience data, particularly in sparse-reward environments, likely because it grounds performance to demonstrated human behavior throughout training.

## 6 DISCUSSION AND CONCLUSION

In this work, we present a novel method for combining behavior cloning with reinforcement learning using an actor-critic architecture that implements a combined loss function and a demonstration-based pre-training phase. We compare our approach against state-of-the-art baselines, including BC, DDPG, and DAPG, and demonstrate the superiority of our method in terms of learning speed, stability, and performance with respect to these baselines. This is shown in the OpenAI Gym LunarLanderContinuous-v2 and the high-fidelity Microsoft AirSim quadrotor simulation environments in both dense and sparse reward settings. This result is especially noticeable in the AirSim landing task (Figure 2c), an environment designed to exhibit a high degree of stochasticity. The BC and DDPG baselines fail to converge to an effective and stable policy after five million training steps on the LunarLanderContinuous-v2 environment with dense reward and the modified version with a sparse reward signal. DAPG, although successful in both LunarLanderContinuous-v2 environments and the custom AirSim landing task, converges at a slower rate when compared to the proposed method and starts the training at a lower performance value after pre-training with demonstration data. Conversely, our method, CoL, is able to quickly achieve high performance without degradation, surpassing both behavior cloning and reinforcement learning algorithms alone, in both dense and sparse reward cases. Additionally, we demonstrate through separate analyses of several components of our architecture that pre-training, the use of a combined loss function, and a fixed ratio of human-generated experience are critical to the performance improvements. This component analysis also indicated that simply sequencing standard behavior cloning and reinforcement learning algorithms does not produce these gains and highlighted the importance of grounding the training to the demonstrated data by using a fixed ratio of expert and agent trajectories in the experience replay buffer.

### 6.1 Future Work

Future work will investigate how to effectively integrate multiple forms of human feedback into an efficient human-in-the-loop RL system capable of rapidly adapting autonomous systems in dynamically changing environments. Actor-critic methods, such as the CoL method proposed in this paper, provide an interesting opportunity to integrate different human feedback modalities as additional learning signals at different stages of policy learning [35]. For example, existing works have shown the utility of leveraging human interventions [7, 26], and specifically learning a predictive model of what actions to ignore at every time step [36], which could be used to improve the quality of the actor's policy. Deep reinforcement learning with human evaluative feedback has also been shown to quickly train policies across a variety of domains [18, 34] and can be a particularly useful approach when the human is unable to provide a demonstration of desired behavior but can articulate when desired behavior is achieved. Further, the capability our approach provides, transitioning from a limited number of human demonstrations to a baseline behavior cloning agent and subsequent improvement through reinforcement learning without significant losses in performance, is largely motivated by the goal of human-in-the-loop learning on physical robotic systems. Thus, our aim is to integrate this method onto such systems and demonstrate rapid, safe, and stable learning from limited human interaction.

## ACKNOWLEDGMENTS

Research was sponsored by the U.S. Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-18-2-0134. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

## REFERENCES


[1] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. 2018. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177* (2018).

[2] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57, 5 (2009), 469–483.

[3] Christopher G Atkeson and Stefan Schaal. 1997. Robot learning from demonstration. In *ICML*, Vol. 97. Citeseer, 12–20.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[5] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* (2015).

[6] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. 2018. Reinforcement Learning from Imperfect Demonstrations. *CoRR* abs/1802.05313 (2018). arXiv:1802.05313 http://arxiv.org/abs/1802.05313

[7] Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich. 2018. Efficiently Combining Human Demonstrations and Interventions for Safe Training of Autonomous Systems in Real-Time. *CoRR* abs/1810.11545 (2018). arXiv:1810.11545 http://arxiv.org/abs/1810.11545

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Jennifer Dy and Andreas Krause (Eds.), Vol. 80. PMLR, Stockholmsmässan, Stockholm Sweden, 1861–1870. http://proceedings.mlr.press/v80/haarnoja18b.html

[9] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. 2018. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[10] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. 2018. Stable Baselines. https://github.com/hill-a/stable-baselines. (2018).

[11] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in neural information processing systems*. 4565–4573.

[12] Sham Kakade. 2001. A Natural Policy Gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic (NIPS'01)*. MIT Press, Cambridge, MA, USA, 1531–1538. http://dl.acm.org/citation.cfm?id=2980539.2980738

[13] Bingyi Kang, Zequn Jie, and Jiashi Feng. 2018. Policy optimization with demonstrations. In *International Conference on Machine Learning*. 2474–2483.

[14] Beomjoon Kim, Amir-massoud Farahmand, Joelle Pineau, and Doina Precup. 2013. Learning from Limited Demonstrations. In *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 2859–2867. http://papers.nips.cc/paper/4918-learning-from-limited-demonstrations.pdf

[15] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[16] W Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the fifth international conference on Knowledge capture*. ACM, 9–16.

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

[18] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L Roberts, Matthew E Taylor, and Michael L Littman. 2017. Interactive learning from policy-dependent human feedback. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2285–2294.

[19] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[20] D Moorhouse and R Woodcock. 1980. US Military Specification MIL–F–8785C. *US Department of Defense* (1980).

[21] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. 2018. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 6292–6299. https://doi.org/10.1109/ICRA.2018.8463162

[22] Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado van Hasselt, John Quan, Mel Vecerík, Matteo Hessel, Rémi Munos, and Olivier Pietquin. 2018. Observe and Look Further: Achieving Consistent Performance on Atari. *CoRR* abs/1805.11593 (2018). arXiv:1805.11593 http://arxiv.org/abs/1805.11593

[23] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. 2018. DAPG for Dexterous Hand Manipulation. https://github.com/aravindr93/hand_dapg. (2018).

[24] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. 2018. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems*. Pittsburgh, Pennsylvania. https://doi.org/10.15607/RSS.2018.XIV.049

[25] S Russell. 1998. Learning agents for uncertain environments (extended abstract). *Conference on Computational Learning Theory (COLT)* (1998), 1–3.

[26] William Saunders, Girish Sastry, Andreas Stuhlmueller, and Owain Evans. 2018. Trial without error: Towards safe reinforcement learning via human intervention. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2067–2069.

[27] Stefan Schaal. 1996. Learning from Demonstration. In *Proceedings of the 9th International Conference on Neural Information Processing Systems (NIPS'96)*. MIT Press, Cambridge, MA, USA, 1040–1046. http://dl.acm.org/citation.cfm?id=2998981.2999127

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized Experience Replay. (2015). arXiv:cs.LG/1511.05952

[29] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2017. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*. arXiv:arXiv:1705.05065 https://arxiv.org/abs/1705.05065

[30] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms.

[31] Richard Sutton and Andrew Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

[32] George E Uhlenbeck and Leonard S Ornstein. 1930. On the theory of the Brownian motion. *Physical review* 36, 5 (1930), 823.

[33] Matej Večerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. 2017. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817* (2017).

[34] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. 2018. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[35] Nicholas R. Waytowich, Vinicius G. Goecks, and Vernon J. Lawhern. 2018. Cycle-of-Learning for Autonomous Systems from Human Interaction. *CoRR* abs/1808.09572v1 (2018). arXiv:1808.09572v1 https://arxiv.org/abs/1808.09572v1

[36] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J Mankowitz, and Shie Mannor. 2018. Learn What Not to Learn: Action Elimination with Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 3562–3573.