REINFORCEMENT LEARNING WITH RANDOM DELAYS

Anonymous authors
Paper under double-blind review

ABSTRACT

Action and observation delays commonly occur in many Reinforcement Learning applications, such as remote control scenarios. We study the anatomy of randomly delayed environments, and show that partially resampling trajectory fragments in hindsight allows for off-policy multi-step value estimation. We apply this principle to derive Delay-Correcting Actor-Critic (DCAC), an algorithm based on Soft Actor-Critic with significantly better performance in environments with delays. This is shown theoretically and also demonstrated practically on a delay-augmented version of the MuJoCo continuous control benchmark.

1 INTRODUCTION

This article deals with the Reinforcement Learning (RL) scenario depicted in Figure 1, which is commonly encountered in real-world applications (Mahmood et al., 2018; Fuchs et al., 2020; Hwangbo et al., 2017). Oftentimes, actions generated by the agent are not immediately applied in the environment, and observations do not immediately reach the agent. Such environments have mainly been studied under the unrealistic (Nilsson et al., 1998; Ge et al., 2013; Mahmood et al., 2018) assumption of constant delays, for which prior work proposed planning algorithms, naively trying to undelay the environment by simulating future observations (Walsh et al., 2008; Schuitema et al., 2010; Firoiu et al., 2018).

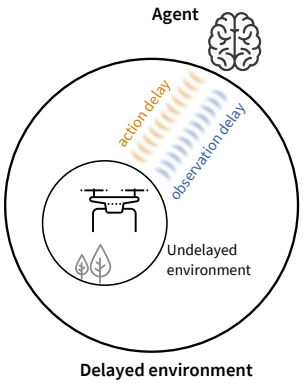

Figure 1: A delayed environment can be decomposed into an undelayed environment and delayed communication dynamics.

We propose an off-policy, planning-free approach that enables low-bias and low-variance multi-step value estimation in environments with random delays. First, we study the anatomy of such environments in order to exploit their structure, defining Random-Delay Markov Decision Processes (RDMDP). Then, we show how to transform trajectory fragments collected under one policy into trajectory fragments distributed according to another policy. We demonstrate this principle by deriving a novel off-policy algorithm (DCAC) based on Soft Actor Critic (SAC), and exhibiting greatly improved performance in delayed environments. Along with this work we release our code, including a wrapper that conveniently augments any OpenAI gym environment with custom delays.

2 DELAYED ENVIRONMENTS

We frame the general setting of real-world Reinforcement Learning in terms of an *agent*, random *observation delays*, random *action delays*, and an *undelayed environment*. At the beginning of each time-step, the agent starts computing a new action from the most recent available delayed observation. Meanwhile, a new observation is sent and the most recent delayed action is applied in the undelayed environment. Real-valued delays are rounded up to the next integer time-step.

For a given delayed observation s_t, the *observation delay* ω_t refers to the number of elapsed time-steps from when s_t finishes being captured to when it starts being used to compute a new action. The *action delay* α_t refers to the number of elapsed time-steps from when the last action influencing s_t starts being computed to one time-step before s_t finishes being captured.

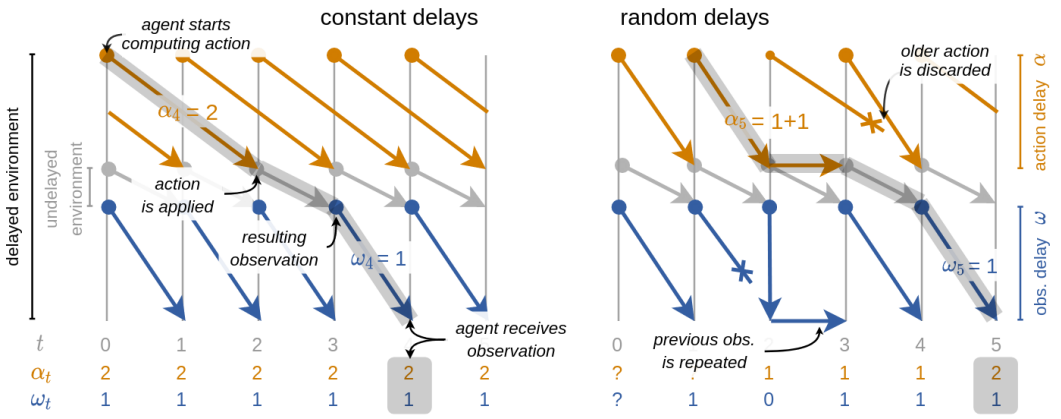

Figure 3: Influence of actions on delayed observations in delayed environments.

We further refer to $\omega_t + \alpha_t$ as the *total delay* of s_t.

As a motivating illustration of real-world delayed setting, we have collected a dataset of communication delays between a decision-making computer and a flying robot over WiFi, summarized in Figure 2. In the presence of such delays, the naive approach is to simply use the last received observation. In this case, any delay longer than one time-step violates the Markov assumption, since the last sent action becomes an unobserved part of the current state of the environment. To overcome this issue, we define a Markov Decision Process that takes into account the communication dynamics.

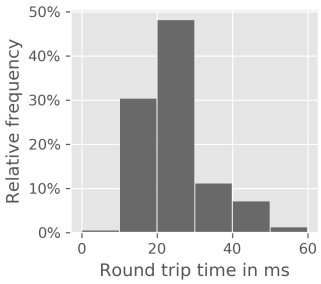

Figure 2: Histogram of real world WiFi delays.

2.1 Random Delay Markov Decision Processes

To ensure the Markov property in delayed settings, it is necessary to augment the delayed observation with at least the last K sent actions. K is the combined maximum possible observation and action delay. This is required as the oldest actions along with the delayed observation describe the current state of the undelayed environment, whereas the most recent actions are yet to be applied (see C.1 Figure 10). Using this augmentation suffices to ensure that the Markov property is met in certain delayed environments. On the other hand, it is possible to do much better when the delays themselves are also part of the state-space. First, this allows us to model self-correlated delays, e.g. discarding outdated actions and observations (see A.1). Second, this provides useful information to the model about how old an observation is and what actions have been applied next (see B.1). Third, knowledge over the total delay allows for efficient credit assignment and off-policy partial trajectory resampling, as we show in this work.

Definition 1. A Random Delay Markov Decision Process $RDMDP(E, p_\omega, p_\alpha) = (X, A, \tilde{\mu}, \tilde{p})$, augments a Markov Decision Process $E = (S, A, \mu, p)$ with:
(1) state space $X = S \times A^K \times \mathbb{N}^2$, (2) action space A,
(3) initial state distribution $\tilde{\mu}(x_0) = \tilde{\mu}(s, u, \omega, \alpha) = \mu(s)\ \delta(u - c_u)\ \delta(\omega - c_\omega)\ \delta(\alpha - c_\alpha)$,
(4) transition distribution $\tilde{p}(s', u', \omega', \alpha', r' | s, u, \omega, \alpha, a) = f_{\omega-\omega'}(s', \alpha', r' | s, u, \omega, \alpha, a) p_\omega(\omega' | \omega) p_u(u' | u, a)$,

where $s \in S$ is the delayed observation, $u \in A^K$ is a buffer of the last K sent actions, $\omega \in \mathbb{N}$ is the *observation delay*, and $\alpha \in \mathbb{N}$ is the *action delay* as defined above. To avoid conflicting with the subscript notation we index the action buffers' elements using square brackets. Here, $u_{[1]}$ is the most recent and $u_{[K]}$ is the oldest action in the buffer. We denote slices by $u_{[i:j]} = (u_{[i]}, \ldots, u_{[j]})$ and $u_{[i:-j]} = (u_{[i]}, \ldots, u_{[K-j]})$. We slightly override this notation and additionally define $u_{[0]} = a$. The constants $c_u \in A^K$ and $c_\omega, c_\alpha \in \mathbb{R}$ initialize u, ω, α, since δ is the Dirac delta distribution. The transition distribution itself is composed of three parts: (1) The observation delay distribution p_ω modelling the evolution of observation delays. Note that this density function must represent a discrete distribution (i.e. be a weighted sum of Dirac delta distributions). Furthermore, this process

will repeat observations if there are no new ones available. This means that the observation delay can maximally grow by one from one time-step to the next. (2) The transition distribution for the action-buffer $p_u(u'|u,a) = \delta(u' - (a, u[1:-1]))$. (3) The distribution f_Δ describing the evolution of observations, rewards and action delays (Definition 2).

Definition 2. For each change in observation delays ($\Delta{=}\omega{-}\omega'$) we define a variable step update distribution f_Δ as

$$f_\Delta(s',\alpha',r'|s,u,\omega,\alpha,a){=}\mathbb{E}_{s^*,\alpha^*,r^*\sim f_{\Delta-1}(\cdot|s,u,\omega,\alpha,a)}[p(s',r'{-}r^*|s^*,u[\overbrace{\omega{-}\Delta}^{\omega'}{+}\alpha'])\ p_\alpha(\alpha'|\alpha^*)]. \quad (1)$$

The base case of the recursion is $f_{-1}(s',\alpha',r' \mid s,u,\omega,\alpha,a) = \delta(s'-s)\ \delta(\alpha'-\alpha)\ \delta(r')$.

Here, p_α is the action delay distribution which, similar to p_ω, must be discrete. The transition distribution of the underlying, undelayed MDP is p. The $r' - r*$ term accumulates intermediate rewards in case observations are skipped or repeated (see Appendix A.2). Since the observation delay cannot increment by more than 1, f_{-1} is used when ω is increasing, whereas f_0 is used when there is no change in observation delay.

A simple special case of the RDMDP is the constant observation and action delay case with $p_\omega(\omega'|\omega) = \delta(\omega' - c_\omega)$ and $p_\alpha(\alpha'|\alpha) = \delta(\alpha' - c_\alpha)$. Here, the RDMDP reduces to a Constantly Delayed Markov Decision Process, described by Walsh et al. (2008). In this case, the action and observation delays α, ω can be removed from the state-space as they don't carry information. Examples of RDMDP dynamics are visualized in Figure 3 (see also Appendix C).

3 Reinforcement Learning in Delayed Environments

Delayed environments as described in Section 2 are specific types of MDP, with an augmented state-space and delayed dynamics. Therefore, using this augmented state-space, traditional algorithms such as Soft Actor Critic (SAC) (Haarnoja et al., 2018a)(Haarnoja et al., 2018b) will always work in randomly delayed settings. However their performance will still deteriorate because of the more difficult credit assignment caused by delayed observations and rewards, on top of the exploration and generalization burdens of delayed environments. We now analyze how to compensate for the credit assignment difficulty by leveraging our knowledge about the delays' dynamics.

One solution is to perform *on-policy* multi-step rollouts on sub-trajectories that are longer than the considered delays. On the other hand, on-policy algorithms are known to be sample-inefficient and therefore are not commonly used in real-world applications, where data collection is costly. This motivates the development of *off-policy* algorithms able to reuse old samples, such as SAC.

Intuitively, in delayed environments, one should take advantage of the fact that actions only influence observations and rewards after a number of time-steps relative to the beginning of their computation (the total delay $\omega + \alpha$). Since the delay information is part of the state-space, it can be leveraged to track the action influence through time. However, applying conventional off-policy algorithms in delayed settings leads to the following issue: the trajectories used to perform the aforementioned multi-step backups have been sampled under an outdated policy, and therefore contain outdated action buffers. In this section, we propose a method to tackle this issue by performing *partial trajectory resampling*. We make use of the fact that the delayed dynamics are known to simulate the effect they would have had under the current policy, effectively transforming off-policy sub-trajectories into on-policy sub-trajectories. This enables us to derive a family of efficient off-policy algorithms for randomly delayed settings.

3.1 Partial Trajectory Resampling in Delayed Environments

One important observation implied by Figure 3 is that, given the delayed dynamics of RDMDPs, some actions contained in the action-buffer of an off-policy state did not influence the subsequent delayed observations and rewards for a number of time-steps. Therefore, if an off-policy sub-trajectory is short enough, it is possible to recursively resample its action buffers with no influence on the return. We propose the following transformation of off-policy sub-trajectories:

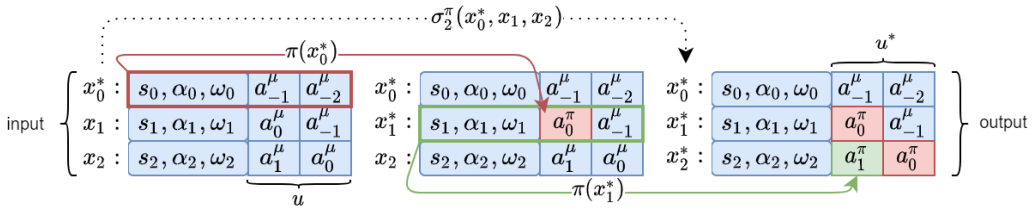

Figure 4: Partial resampling of a small sub-trajectory. The action buffer is recursively resampled according to the current policy π (rewards are not modified by σ and are omitted here).

Definition 3. The partial trajectory resampling operator recursively updates action buffers as follows

$$\sigma_n^\pi(\underbrace{s_1^*, u_1^*, \omega_1^*, \alpha_1^*}_{x_1^*}, r_1^*, \tau_{n-1}^* | x_0^*; \underbrace{s_1, u_1, \omega_1, \alpha_1}_{x_1}, r_1, \tau_{n-1})$$
$$= \delta((s_1^*, \omega_1^*, \alpha_1^*, r_1^*) - (s_1, \omega_1, \alpha_1, r_1)) \mathbb{E}_{a_0 \sim \pi(\cdot|x_0^*)}[\delta(u_1^* - (a_0, u_0^*[1:-1]))]\, \sigma_{n-1}^\pi(\tau_{n-1}^* | x_1^*; \tau_{n-1}) \quad (2)$$

with trivial base case $\sigma_0(x_0^*) = 1$

This operator recursively resamples the most recent actions of each action-buffer in an input sub-trajectory τ_n, according to a new policy π. Everything else stays unchanged. A visual example is provided in Figure 4 with $n = 2$ and an action buffer of 2 actions. When resampled actions are delayed and would not affect the environment, they do not "invalidate" the sub-trajectory. The resampled trajectories can then be considered on-policy.

Theorem 1. The partial trajectory resampling operator σ_n^π (Def. 3) transforms off-policy trajectories into on-policy trajectories

$$\mathbb{E}_{\tau_n \sim p_n^\mu(\cdot|x_0)}[\sigma_n^\pi(\tau_n^* | x_0; \tau_n)] = p_n^\pi(\tau_n^* | x_0) \quad (3)$$

on the condition that none of the delayed observations depend on any of the resampled actions, i.e.

$$\omega_t^* + \alpha_t^* \geq t \quad (4)$$

where t indexes the trajectory $\tau_n^* = (s_1^*, u_1^*, \omega_1^*, \alpha_1^*, r_1^*, \ldots, s_n^*, u_n^*, \omega_n^*, \alpha_n^*, r_n^*)$ from 1 to n.

The condition in Equation 4 can be understood visually with the help of Figure 3. In the constant delay example it is fulfilled until the third timestep. After that the observations would have been influenced by the resampled actions (starting with a_0).

3.2 Multistep Off-Policy Value Estimation in Delayed Environments

We have shown in Section 3.1 how it is possible to transform off-policy sub-trajectories into on-policy sub-trajectories in the presence of random delays. From this, we can derive a family of efficient off-policy algorithms for the randomly delayed setting. For this matter, we make use of the classic on-policy Monte-Carlo n-step value estimator:

Definition 4. The n-step state-value estimator is defined as

$$\hat{v}_n(x_0; \underbrace{x_1^*, r_1^*, \tau_{n-1}^*}_{\tau_n^*}) = r_1^* + \gamma \hat{v}_{n-1}(x_1^*; \tau_{n-1}^*) = \sum_{i=1}^{n} \gamma^{i-1} r_i^* + \gamma^n \hat{v}_0(x_n^*). \quad (5)$$

where $\hat{v}_0$ is a state-value function approximator (e.g. a neural network).

Indeed, in γ-discounted RL, performing on-policy n-step rollouts to estimate the value function reduces the bias introduced by the function approximator by a factor of γ^n:

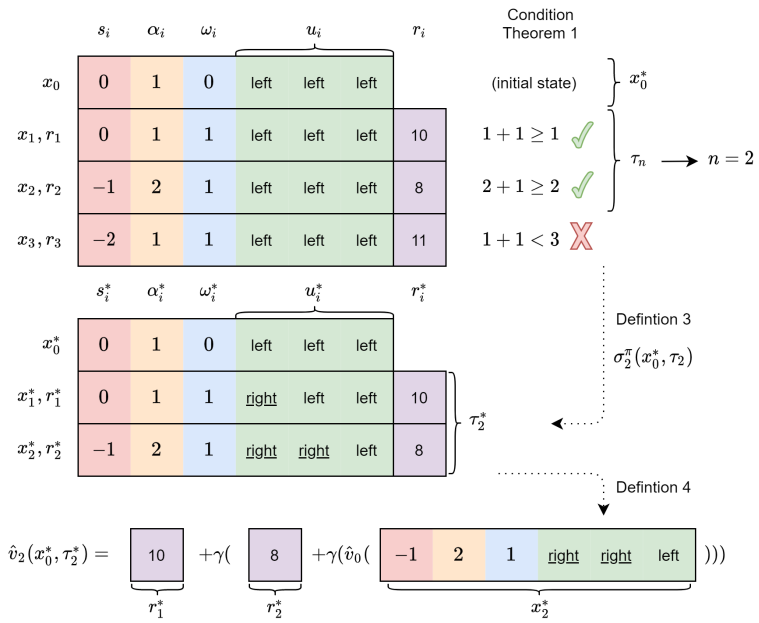

Figure 5: Visual example in a 1D-world with random delays ($K = 3$). The original trajectory has been sampled under the policy μ: 'always go left'. The current policy is π: 'always go right'.

Lemma 1. The n-step value estimator has the following bias:

$$\text{bias}(\hat{v}_n(x_0, \cdot)) = \gamma^n \mathbb{E}_{\ldots, x_n^*, r_n^* \sim p_n^\pi(\cdot|x_0)}[\text{bias}(\hat{v}_0(x_n^*))] \tag{6}$$

A simple corollary of Lemma 1 is that the on-policy n-step value estimator is unbiased when the function approximator $\hat{v}_0$ is unbiased. On the other hand, Theorem 1 provides a recipe for transforming sub-trajectories collected under old policies into actual on-policy sub-trajectories. From a given state in an off-policy trajectory, this is done by applying σ_n^π to all the subsequent transitions until we meet a total delay $(\omega_i + \alpha_i)$ that is shorter than the length of the formed sub-trajectory. Consequently, the transformed sub-trajectory can be fed to the on-policy n-step value estimator, where n is the length of this sub-trajectory. Not only this gives a better value estimate than usual 1-step off-policy estimators according to Lemma 1, but this maximally compensates for the multi-step credit assignment difficulty introduced by random delays. Indeed, the length of the transformed sub-trajectory is then exactly the number of time-steps it took the first action of the sub-trajectory to have an influence on subsequent delayed observations, minus one time-step.

As opposed to other unbiased n-step off-policy methods such as importance sampling and Retrace (Munos et al., 2016), this method doesn't suffer from variance explosion. This is because the presence of delays allows us to transform off-policy sub-trajectories into on-policy sub-trajectories, so that old samples don't need to be weighted by the policy ratio.

Although we use a multi-step state-value estimator, the same principles can be applied to action-value estimation as well. In fact, the trajectory transformation described in Definition 3 enables efficient off-policy n-step value estimation in any value-based algorithm that would otherwise perform 1-step action-value backups, such as DQN, DDPG or SAC. In the next section, we illustrate this using SAC.

Figure 5 summarizes the whole procedure in a simple 1D-world example. The maximum possible delay is $K = 3$ here, and the agent can only go 'left' or 'right'. An initial augmented state x_0 is sampled from the replay memory, along with the 3 subsequent augmented states and rewards. The condition of Theorem 1 is satisfied for $n \leq 2$. It follows that $\tau_n = \tau_2 = (x_1, x_2)$. This off-policy trajectory fragment is partially resampled, which yields the corresponding on-policy trajectory fragment $\tau_n^* = \tau_2^*$. This on-policy trajectory fragment can then be used to compute an unbiased n-step value estimate of the initial state $x_0 = x_0^*$.

4 DELAY-CORRECTING ACTOR-CRITIC

We have seen in Section 3 how it is possible, in the delayed setting, to collect *off-policy* trajectories and still use *on-policy* multi-step estimators in an unbiased way, which allows us to compensate for the more difficult credit assignment introduced by the presence of random delays. We now apply this method to derive Delay-Correcting Actor-Critic (DCAC), an improved version of Soft Actor-Critic (Haarnoja et al., 2018a;b) for real-time randomly delayed settings.

4.1 VALUE APPROXIMATION

Like SAC, DCAC makes use of the entropy-augmented soft value function (Haarnoja et al., 2018a):

Lemma 2. In a RDMDP (E, p_ω, p_α) the soft value function is:

$$v^{\text{soft}}(x_0^*) = \mathbb{E}_{a\sim\pi(\cdot|x_0^*)}[\mathbb{E}_{x_1^*,r_1^*\sim\tilde{p}(\cdot|x_0^*,a)}[r_1^* + \gamma v^{\text{soft}}(x_1^*)] - \log\pi(a|x_0^*)] \tag{7}$$

It can be estimated by augmenting the reward function in Definition 4 with an entropy reward:

Definition 5. The delayed on-policy n-step soft state-value estimator, i.e. the n-step state-value estimator with entropy augmented rewards under the current policy π, is

$$\hat{v}_n^{\text{soft}}(x_0^*; \tau_n^*) = r_1^* + \gamma \hat{v}_{n-1}^{\text{soft}}(x_1^*; \tau_{n-1}^*) - \mathbb{E}_{a\sim\pi(\cdot|x_0^*)}[\log\pi(a|x_0^*)] \tag{8}$$

where $\hat{v}_0^{\text{soft}}$ is a state-value function approximator (e.g. a neural network).

Given the off-policy trajectory transformation proposed in Section 3, Definition 5 directly gives DCAC's value target. To recap, we sample an initial state x_0 $(= x_0^*)$ and a subsequent trajectory τ_n $(= x_1, r_1, \dots x_n, r_n)$ from a replay memory. The sampling procedure ensures that n is the largest length such that the sampled trajectory τ_n does not contain any total delay $\omega_i + \beta_i < i$. This trajectory was collected under an *old policy* μ, but we need a trajectory compatible with the *current policy* π to use $\hat{v}_n^{\text{soft}}$ in an unbiased way. Therefore, we feed τ_n to the partial trajectory resampling operator defined in Definition 3. This produces an *equivalent on-policy* sub-trajectory τ_n^* with respect to the current policy π according to Theorem 1, while maximally taking advantage of the bias reduction described by Lemma 1. This partially resampled on-policy sub-trajectory is fed as input to $\hat{v}_n^{\text{soft}}(x_0; \tau_n^*)$, which yields the target used in DCAC's soft state-value loss:

Definition 6. The DCAC critic loss is

$$L_v^{\text{DCAC}}(v) = \mathbb{E}_{(x_0,\tau_n)\sim\mathcal{D}}\, \mathbb{E}_{\tau_n^*\sim\sigma_n^\pi(\cdot|x_0;\tau_n)}[(v_\theta(x_0) - \hat{v}_n^{\text{soft}}(x_0;\ \tau_n^*))^2] \tag{9}$$

where x_0, τ_n are a start state and following trajectory, sampled from the replay memory, and satisfying the condition of Theorem 1.

4.2 POLICY IMPROVEMENT

In addition to using the on-policy n-step value estimator as target for our parametric value estimator, we can also use it for policy improvement. As in SAC we use the reparameterization trick (Kingma & Welling, 2013) to obtain the policy gradient from the value estimator. However, since we use our trajectory transformation and a multi-step value estimator, this involves backpropagation through time in the action-buffer.

Definition 7. The DCAC actor loss is

$$L_\pi^{\text{DCAC}}(\pi) = -\mathbb{E}_{(x_0,\tau_n)\sim\mathcal{D}}\, \mathbb{E}_{\tau_n^*\sim\sigma_n^\pi(\cdot|x_0;\tau_n)}[\hat{v}_n^{\text{soft}}(x_0;\ \tau_n^*)] \tag{10}$$

where x_0, τ_n are a start state and following trajectory, sampled from the replay memory, and satisfying the condition of Theorem 1.

Proposition 1. The DCAC actor loss is a less biased version of the SAC actor loss with

$$\text{bias}(L_\pi^{\text{DCAC}}) = \mathbb{E}_n[\gamma^n]\,\text{bias}(L_\pi^{\text{SAC}}) \tag{11}$$

assuming both are using similarly biased parametric value estimators to compute the loss, i.e.

$$\text{bias}(\hat{v}_0^{\text{soft}}(x)) = \mathbb{E}_{a\sim\pi(\cdot|x)}[\text{bias}(\hat{q}_0^{\text{soft}}(x,a))] \tag{12}$$

5 EXPERIMENTAL RESULTS

To evaluate our approach and make future work in this direction easy for the RL community, we release as open-source, along with our code, a Gym wrapper that introduces custom multi-step delays in any classical turn-based Gym environment. In particular, this enables us to introduce random delays to the Gym MuJoCo continuous control suite (Brockman et al., 2016; Todorov et al.), which is otherwise turn-based.

Compared algorithms. A naive version of SAC would only use the unaugmented delayed observations, which violates the Markov assumption in delayed settings as previously pointed out. Consequently, naive SAC exhibits near-random results in delayed environments. A few such experiments are provided in the Appendix for illustration (Figure 12).

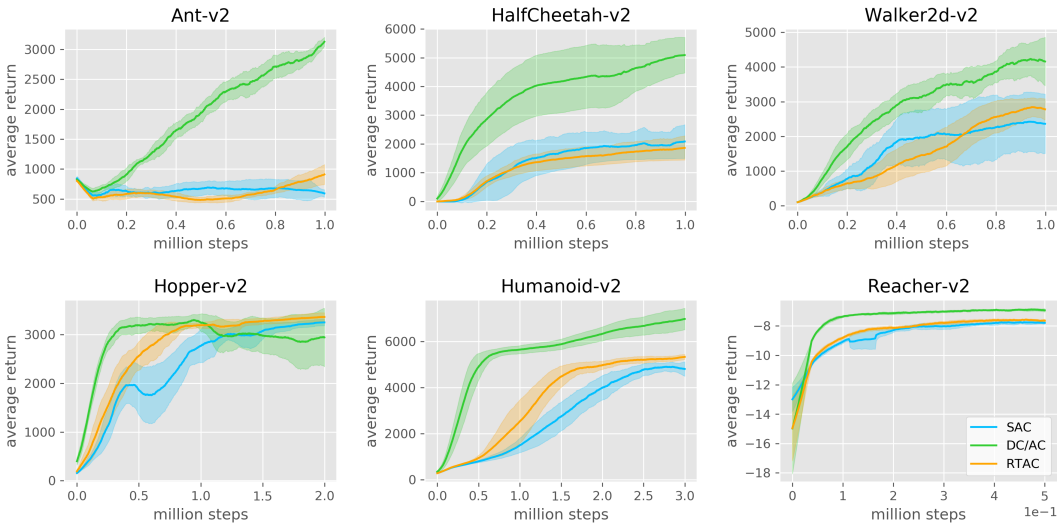

Figure 6: $\omega = 2, \alpha = 3$ (constant delays). With a constant total delay of 5 time-steps, DCAC exhibits a very strong advantage in performance. All tested algorithms use the same RDMDP augmented observations.

In order to make a fair comparison, all other experiments compare DCAC against SAC in the same RDMDP setting, i.e. all algorithms use the augmented observation space defined in Section 2.1. Since SAC is the algorithm we chose to improve for delayed scenarios, comparing DCAC against it in the same setting provides a like-for-like comparison. We also found interesting to compare against RTAC (Ramstedt & Pal, 2019). Indeed, DCAC reduces to this algorithm in the special case where observation transmission is instantaneous (ω=0) and action computation and transmission constantly takes 1 time-step (α=1). Whereas DCAC performs variable-length state-value backups with partial trajectory resampling as explained in Section 4 , RTAC performs 1-step state-value backups, and SAC performs the usual 1-step action-value backup described in its second version (Haarnoja et al., 2018b). All hyperparameters and implementation details are provided in Section B of the Appendix.

For each experiment, we perform six runs with different seeds, and shade the 90% confidence intervals.

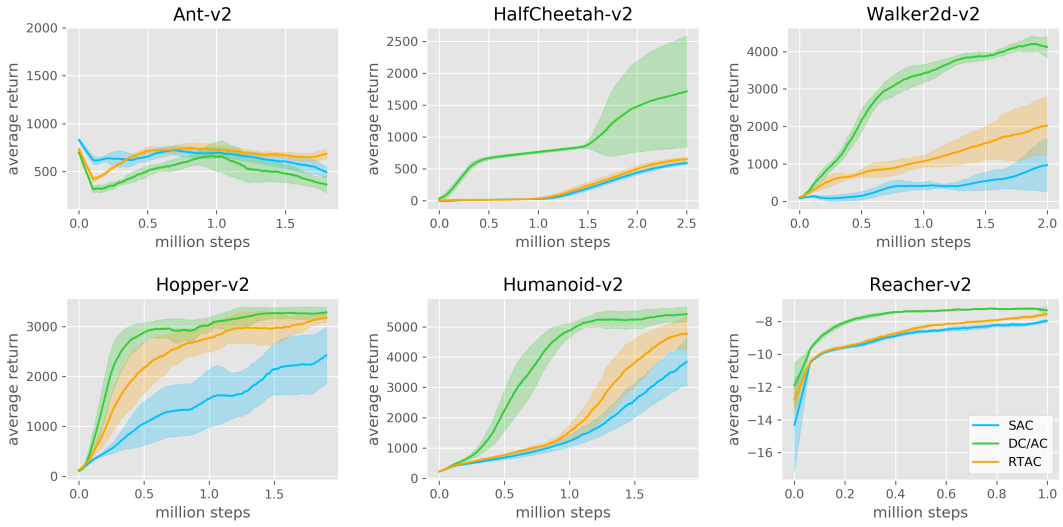

Figure 7: $\alpha, \omega \sim$ WiFi (random delays). DCAC clearly dominates the baselines. Ant became too difficult for all tested algorithms. HalfCheetah also became difficult and only DCAC escapes from local minima.

Constant delays. Our first batch of experiments features simple, constantly delayed scenarios. Figure 6 displays the results of the most difficult of these experiments (i.e. where the delays are longest), while the others are provided in Section D.2 of the Appendix. The advantage of using DCAC is obvious in the presence of long constant delays. Note that DCAC reduces to the RTAC (Ramstedt & Pal, 2019) algorithm when $\omega = 0$ and $\alpha = 1$, and behaves as an evolved form of RTAC in the presence of longer constant delays.

Real-world random delays. Our second batch of experiments features random delays of different magnitudes. The experiment we chose to present in Figure 7 is motivated by the fact that our approach is designed for real-world applications. Importantly, it provides an example how to implement DCAC in practice (see Appendix A and B for more details). We sample the communication delays for actions and observations from our real-world WiFi dataset, presented in Figure 2. When action or observation communications supersede previous communications, only the most recently produced information is kept. In other words, when an action is received in the undelayed environment, its age is compared to the action that is currently being applied. Then, the one that the agent most recently started to produce is applied. Similarly, when the agent receives a new observation, it only keeps the one that was most recently captured in the undelayed environment (see the right-hand side of Figure 3 for a visual example). We discretize the communication delays by using a time-step of 20ms. Importantly, note that Figure 2 has been cropped to 60ms, but the actual dataset contains outliers that can go as far as 1s. However, long delays (longer than 80ms in our example) are almost always superseded and discarded. Therefore, when such information is received, we clip the corresponding delay with no visible impact in performance: in practice, the maximum acceptable delays are design choices, and can be guided by existing probabilistic timing methods (Santinelli et al., 2017).

6 Related work

We trace our line of research back to Katsikopoulos & Engelbrecht (2003), who provided the first discussion about Delayed Markov Decision Processes. In particular, they were interested in asynchronous rewards, which provides interesting insights in relation to Appendix A.2. Walsh et al. (2008) later re-introduced the notion of "Constantly Delayed Markov Decision Process". While recent advances in deep learning enable implementations of what the authors call an "augmented approach", this was considered intractable at the time because the size of the action buffer grows with the considered delay length. Instead, they studied the case where observations are retrieved with a constant delay and developed a model-based algorithm to predict the current state of the environment. Similarly, Schuitema et al. (2010) developed "memory-less" approaches based on SARSA and vanilla

Q-learning, taking advantage of prior knowledge about the duration of a constant control delay. Hester & Stone (2013) adopted the action-buffer-augmented approach to handle random delays, and relied on a decision-tree algorithm to perform credit assignment implicitly. By comparison, our approach relies on delay measurements to perform credit assignment explicitly. More recently, Firoiu et al. (2018) introduced constant action delays to a video game to train agents whose reaction time compares to humans. Similar to previous work, the authors used a state-predictive model, but based on a recurrent neural network architecture. Ramstedt & Pal (2019) formalized the framework of Real-Time Reinforcement Learning (RTRL) that we generalize here to all forms of real-time delays. Initially designed to cope with the fact that inference is not instantaneous in real-world control, the RTRL setting is equivalent to a constantly delayed MDP with $\alpha = 1$ and $\omega = 0$. Finally, Xiao et al. (2020) adopted an alternative approach by considering the influence of the action selection time when action selection is performed within the duration of a larger time-step. However their framework only allows delays smaller than one time-step, whereas large time steps are not compatible with high-frequency control.

7 Conclusion and future work

We proposed a deep off-policy and planning-free approach that explicitly tackles the credit assignment difficulty introduced by real-world random delays. This is done by taking advantage of delay measurements in order to generate actual on-policy sub-trajectories from off-policy samples.

In addition, we provide a theoretical analysis that can easily be reused to derive a wide family of algorithms such as DCAC, while previous work mostly dealt with finding approximate ways of modelling the state-space in constantly delayed environments. The action-buffer is fundamentally required to define a Markovian state-space for RDMDPs, but it is of course possible to observe this action-buffer approximately, e.g. by compressing it in the hidden state of an RNN, which is complementary to our work. We have designed our approach with real-world applications in mind, and it is easily scalable to a wide variety of scenarios (for a practical implementation, see Section 5 and Sections A and B of the Appendix).

To the best of our knowledge, DCAC is the first deep actor-critic approach to exhibit such strong performance on both randomly and constantly delayed settings, as it makes use of the partially known dynamics of the environment to compensate for difficult credit assignment. We believe that our model can be further improved by making use of the fact that our critic estimates the state-value instead of the action-value function. Indeed, in this setting, Ramstedt & Pal (2019) showed that it is possible to simplify the model by merging the actor and the critic networks using the PopArt output normalization (van Hasselt et al., 2016), which we did not try yet and leave for future work.

Our approach handles and adapts to arbitrary choices of time-step duration, although in practice time-steps smaller than the upper bound of the inference time will require a few tricks. We believe that this approach is close to time-step agnostic RL, and will investigate this direction in future work.

REFERENCES

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

Vlad Firoiu, Tina Ju, and Joshua B. Tenenbaum. At human speed: Deep reinforcement learning with action delay. *CoRR*, abs/1810.07286, 2018.

Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Duerr. Super-human performance in gran turismo sport using deep reinforcement learning, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Yuan Ge, Qigong Chen, Ming Jiang, and Yiqing Huang. Modeling of random delays in networked control systems. *Journal of Control Science and Engineering*, 2013, 2013.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Todd Hester and Peter Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013.

Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, 2017.

Konstantinos V Katsikopoulos and Sascha E Engelbrecht. Markov decision processes with delays and asynchronous cost collection. *IEEE transactions on automatic control*, 48(4):568–574, 2003.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

A. Rupam Mahmood, Dmytro Korenkevych, Brent J. Komer, and James Bergstra. Setting up a reinforcement learning task with a real-world robot, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Remi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 1054–1062. Curran Associates, Inc., 2016.

Johan Nilsson, Bo Bernhardsson, and Björn Wittenmark. Stochastic analysis and control of real-time systems with random time delays. *Automatica*, 34(1):57–64, 1998.

Simon Ramstedt and Christopher Pal. Real-time reinforcement learning. In *NeurIPS*, 2019.

L. Santinelli, F. Guet, and J. Morio. Revising measurement-based probabilistic timing analysis. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 199–208, 2017.

Erik Schuitema, Lucian Busoniu, Robert Babuska, and Pieter Jonker. Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach. In *International Conference on Intelligent Robots and Systems*, 2010.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. in 2012 ieee. In *RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

Hado P van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, pp. 4287–4295, 2016.

Thomas J. Walsh, Ali Nouri, Lihong Li, and Michael L. Littman. Learning and planning in environments with delayed feedback. *Autonomous Agents and Multi-Agent Systems*, 18:83–105, 2008.

Ted Xiao, Eric Jang, Dmitry Kalashnikov, Sergey Levine, Julian Ibarz, Karol Hausman, and Alexander Herzog. Thinking while moving: Deep reinforcement learning with concurrent control. In *International Conference on Learning Representations*, 2020.

A PRACTICAL CONSIDERATIONS AND SCALABILITY

A.1 NOTE ON RDMDPS

The way we chose to define RDMDP s, the agent only keeps the delayed observation that was most recently captured in the undelayed environment. Ideally, it is also ensured by the undelayed environment that the applied action is the action that most recently started being computed by the agent. In practice, this can be ensured by augmenting the actions with time-stamps corresponding to the beginning of their computation, and observations with time-stamps corresponding to the end of their capture. Thus, the undelayed environment and the agent can keep track of the most recent received time-stamp and discard outdated incoming information.

The separation between ω and α is mainly required to allow auto-correlated conditional distributions on both delays, so for example superseded actions and observations can be discarded. In such situations, it is optimal to observe ω and α (or more exactly ω and κ, see Sections B.1 and B.4) separately. However this might not always be possible in practice. Note that DCAC doesn't make use of the separate delays, and instead simply looks at the total delays ($\omega_i + \alpha_i$). Therefore, when only the total delays are available (they can easily be measured by the agent), they can still be used by DCAC and observed by the model.

As seen in our WiFi experiment, the maximum delays are design choices in practice, since the actual maximum delays can be prohibitively long and would require a long action buffer to be handled in the worst-case scenario. Observations reaching the agent with a total delay that exceeds the chosen K value should simply be discarded, and a procedure implemented to handle the unlikely edge-case where more than K such observations are received in a row.

Finally, the action buffer (u) is required to define the mathematical framework as a Markov Decision Process. When a small time-step is chosen, the size of the action-buffer grows large. One possibility to practically handle such situation if desired is to compress the action buffer in the hidden state of an RNN, so the model observes it in an approximate manner.

A.2 NOTE ON DELAYED REWARDS

We have implicitly made a choice when defining the rewards for RDMDP s. Indeed, keep in mind that observations can be dropped (superseded) at the level of the agent. In such cases, we chose to accumulate the rewards corresponding to the lost transitions. When an observation gets repeated because no new observation is available, the corresponding reward is 0, and when a new observation arrives, the corresponding reward contains the sum of intermediate rewards in lost transitions.

In practice, this is ensured for example by making the assumption that the remote robot (i.e. the undelayed environment) can observe its own instantaneous reward. This allows the robot to compute its cumulative reward and send it to the agent along with the observation. The agent can then compute the difference between the last cumulative reward it received from the remote robot and the new one for each incoming observation (NB: outdated observations are discarded so the agent only sees cumulative rewards with time-increasing timestamps).

Alternatively, the practitioner can choose to repeat the delayed rewards along with the repeated delayed observations at the level of the agent (this is what we use to do in earlier versions of the paper). When a trick similar to the aforementioned cannot be implemented, this can be done instead, with no impact on our analysis. However, the reward signal will inherently have a higher variance.

A.3 LONG OBSERVATION CAPTURE OR COMBINED OBSERVATIONS

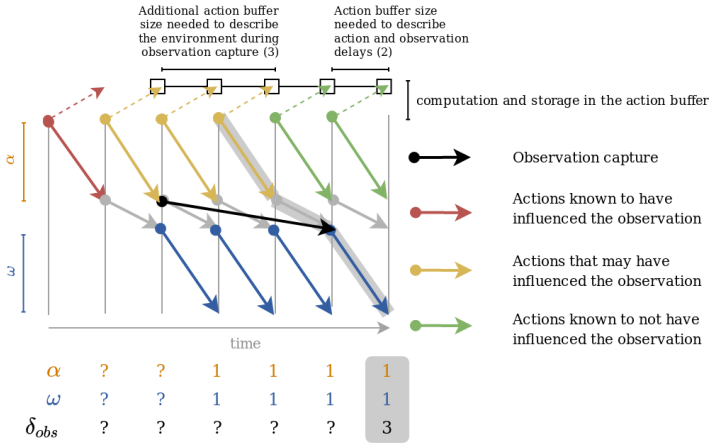

Figure 8: When observation capture is not instantaneous, the size of the action buffer must be increased.

In this work, we have implicitly assumed that observation capture is instantaneous whereas processing and transmission can be part of the observation delay. However, in practice, it is often the case that observation capture cannot be considered instantaneous. In such situations, one should increase the size of the action buffer by a number of time-steps corresponding to the maximum duration of observation capture, and append the observation capture duration δ_{obs} to the observation when it is available and not constant. Indeed, as illustrated in Figure 8, when observation capture is not instantaneous it is not possible to know which undelayed state(s) it describes. The length of the multi-step backup performed by DCAC doesn't need to be adapted, because it only cares about the first action that is known to not have influenced the delayed observation.

The same holds when observations are formed of several combined parts that were captured at different timestamps. In such situation, one can consider the whole span of these timestamps as the observation capture duration. Additionally, these timestamps can be appended to the formed observation when they are available.

B IMPLEMENTATION DETAILS

B.1 A BETTER ACTION DELAY FOR THE MODEL

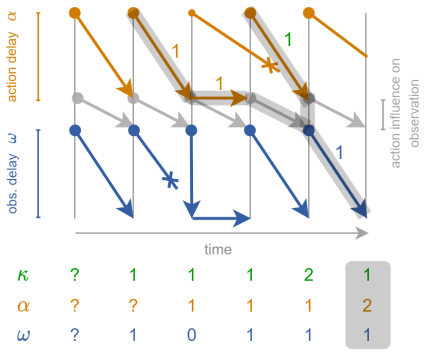

Figure 9: In a RDMDP , κ gives more information than α to the model, because it identifies the last action that is known to be applied when the observation is sent from the undelayed environment.

The action delay α is needed to define RDMDP s, and is used by DCAC. It is also useful as an input to the model, because it identifies the last action that influenced the observation, and therefore gives

information about what actions may be applied next. However, in RDMDP s, this information is not complete. We do better by appending another kind of action delay to the observations on top of α, which we call κ. κ identifies the action that is going to be applied in the undelayed environment when the captured observation is sent, whereas α identifies the action that was applied during the previous time-step. We use κ instead of α as input of the model for all tested algorithms.

B.2 Model Architecture

The model we use in all our experiments is composed of two separate multi-layer perceptrons (MLPs): a critic network, and an actor network. Both MLPs are built with the same simple architecture of 2 hidden layers, 256 units each. The critic outputs a single value, whereas the actor outputs an action distribution with the dimension of the action space, from which actions are sampled with the reparameterization trick. This architecture is compatible with the second version of SAC described in Haarnoja et al. (2018b). The only difference from the DCAC model is that the SAC critic tracks $q(x)$, and not $v(x)$. Indeed, differently from usual actor-critic algorithms, the output of DCAC's critic approximates the state-value $v(x)$ (instead of the action-value $q(x)$), as it is sufficient to optimize the actor loss described in Definition 7. Weights and biases are initialized with the default Pytorch initializer. Both the actor and the critic are optimized by gradient descent with the Adam optimizer, on losses $L^{\text{DCAC}}(\pi)$ (equation 10) and $L^{\text{DCAC}}(v)$ (equation 9), respectively. Classically, we use twin critic networks (Van Hasselt et al., 2015; Fujimoto et al., 2018) with target weight tracking (Mnih et al., 2015) to stabilize training.

B.3 Constant delays

In the constantly delayed setting, the delays ω and α are fixed, and can be implicitly considered as a constant property of the environment. In this setting, we simply concatenate the members s and u of the augmented observation $x = s, u, \omega, \alpha$, and feed the concatenation to the model.

B.4 RDMDPs

To handle the more general case of $RDMDP(E, p_\alpha, p_\omega)$, we need the $\omega_i + \alpha_i$ information for each delayed observation s_i. This is achieved by appending an identifier to the action computed by the agent. The identifier of the action that was applied during the time-step prior to the end of observation capture is then sent back to the agent, along with the observation. When the agent receives the identifier back, it looks the identified action up in the action-buffer. This action is found at the $(\omega_i + \alpha_i)^{th}$ position, as it was sent $\omega_i + \alpha_i$ time-steps ago.

The action identifier can always be the timestamp at which the agent started computing the action. Making the additional assumption that the remote robot and the agent have synchronized clocks, the robot can directly append to the observation the local timestamp corresponding to the end of observation capture and this timestamp can be used by the agent to deduce ω_i, κ_i and α_i separately.

Although this is not useful for the training algorithm (which only needs the $\omega_i + \alpha_i$ information), this is interesting as inputs of the model for inference, because this adds the information 'how old is the delayed observation' (ω_i) to the information 'how old is the action that influenced it' ($\omega_i + \alpha_i$).

Thus we feed the concatenation of x_i, ω_i and κ_i (c.f. Section B.1) as input of our model for all tested algorithms. More exactly, we feed ω_i and κ_i as one-hot encoded inputs, since we believe this maps to positions in the action-buffer (i.e. to past moments in time) in a more natural way than feeding their raw integer values.

The aforementioned details are implemented for all tested algorithms, as they are related to RDMDP s rather than DCAC in particular.

B.5 Hyperparameters

Other than our neural network architecture, our implementations of SAC, RTAC and DCAC all share the following hyperparameters:

Table 1: Hyperparameters

Name	Value
Optimizer	Adam (Kingma & Ba, 2014)
Learning rate	0.0003
Discount factor (γ)	0.99
Batch size	128
Target weights update coefficient (τ)	0.005
Gradient steps / environment steps	1
Reward scale	5.0
Entropy scale	1.0
Replay memory size	1000000
Number of samples before training starts	10000
Number of critics	2

NB: the target weights are updated according to the following running average: $\overline{\theta} \leftarrow \tau\theta + (1-\tau)\overline{\theta}$

B.6 MINIBATCH GRADIENT DESCENT

As the reader may have noticed, a difficulty arises when implementing DCAC with minibatch tensors: with random delays, the length of the performed multi-step backup is different for each individual trajectory sampled from the replay memory. In our implementation, we alleviate this by sampling trajectories of length K. We then compute a mask in a batched fashion, representing for each trajectory the longest sub-trajectory that satisfies the condition of Theorem 1. This mask is then used to resample the action-buffers and perform the multi-step backup in a batched fashion. For more details, we refer the reader to our code.

C VISUAL EXAMPLES

C.1 CONSTANT DELAYS

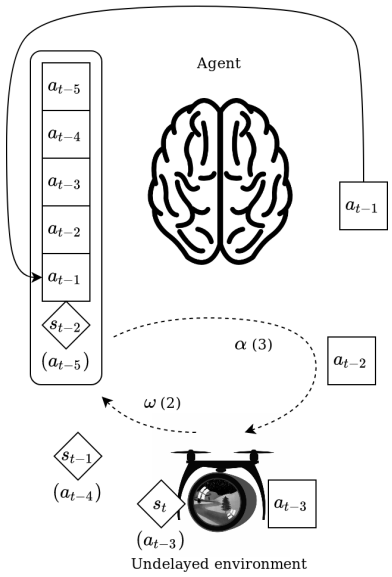

Figure 10: Visual example of a Constantly Delayed MDP, with an action delay of 3 time-steps and an observation delay of 2 time-steps. Here, actions are indexed by the time at which they started being produced. The augmented observation is composed of an action-buffer of the last 5 computed actions along with the delayed observation s_{t-2}. It will be used by the agent to compute action a_t. Meanwhile, in the undelayed environment, action a_{t-3} is received and observation s_t is captured.

C.2 RANDOM DELAYS

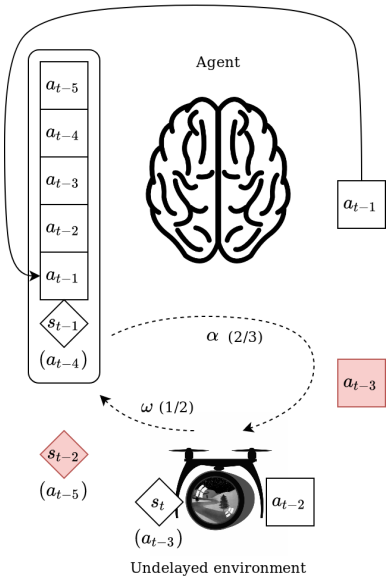

Figure 11: Visual example of a Random Delay MDP, with $\alpha \leq 3$ time-steps and $\omega \leq 2$ time-steps. Actions and observations may be superseded due to random delays. In such cases, only the most recently produced actions and observations are kept, the others (in red) are discarded.

D ADDITIONAL EXPERIMENTS

D.1 IMPORTANCE OF THE AUGMENTED OBSERVATION SPACE

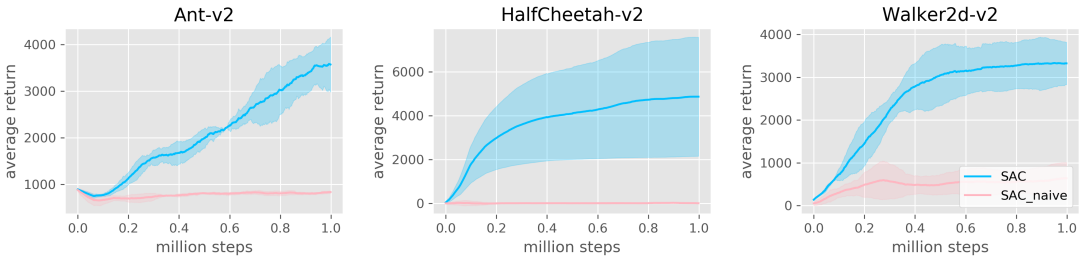

Figure 12: $\omega = 0, \alpha = 1$: We illustrate the importance of the augmented observation space in delayed settings using our simplest task (constant 1-step action delay). Even with this small 1-step constant delay, the delayed observations are not Markov and a naive algorithm using only these observations (here: SAC naive) has near-random results. By comparison, an algorithm using the $RDMDP$ augmented observations instead (here: SAC) is able to learn in delayed environments.

D.2 CONSTANT DELAYS

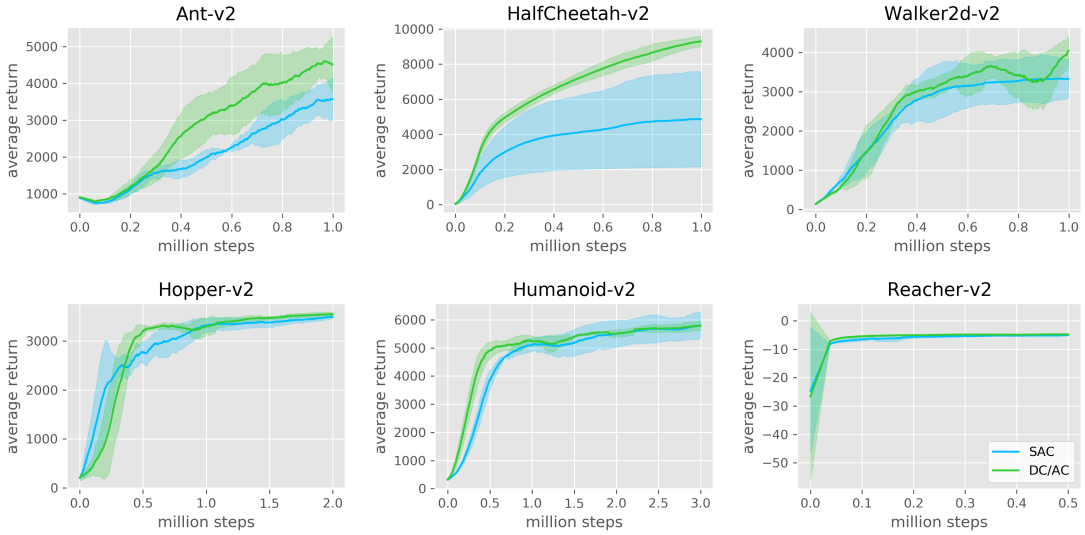

Figure 13: $\omega = 0, \alpha = 1$: This specific setting is equivalent to the RTRL setting (Ramstedt & Pal, 2019), in which DCAC reduces to the vanilla RTAC algorithm (without output normalization and merged networks). DCAC (RTAC) slightly outperforms SAC in this setting.

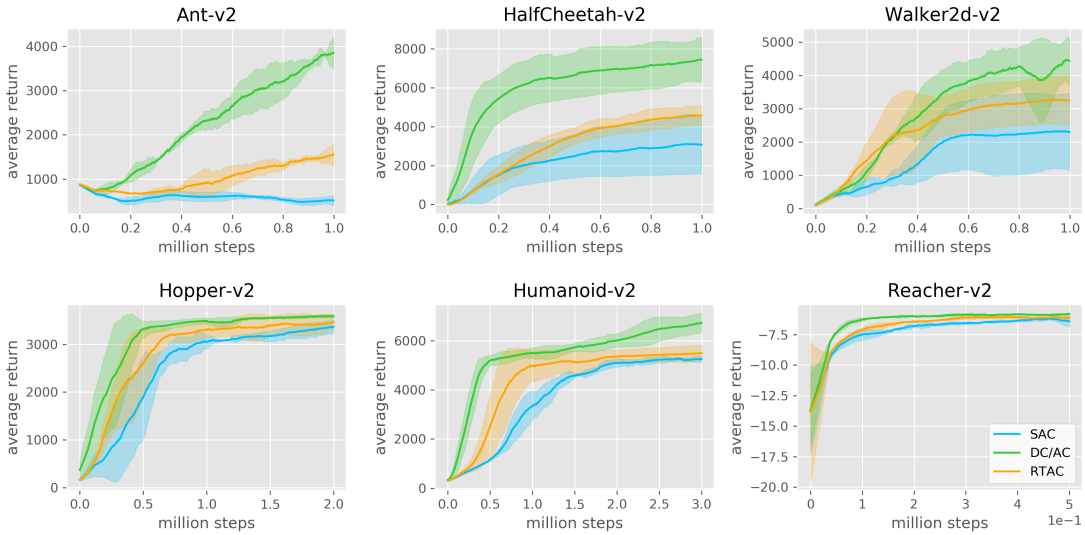

Figure 14: $\omega = 1, \alpha = 2$: In this more difficult setting (total constant delay of 3 instead of 1), DCAC starts really showing its potential, clearly outperforming all other approaches.

D.3 RANDOM DELAYS

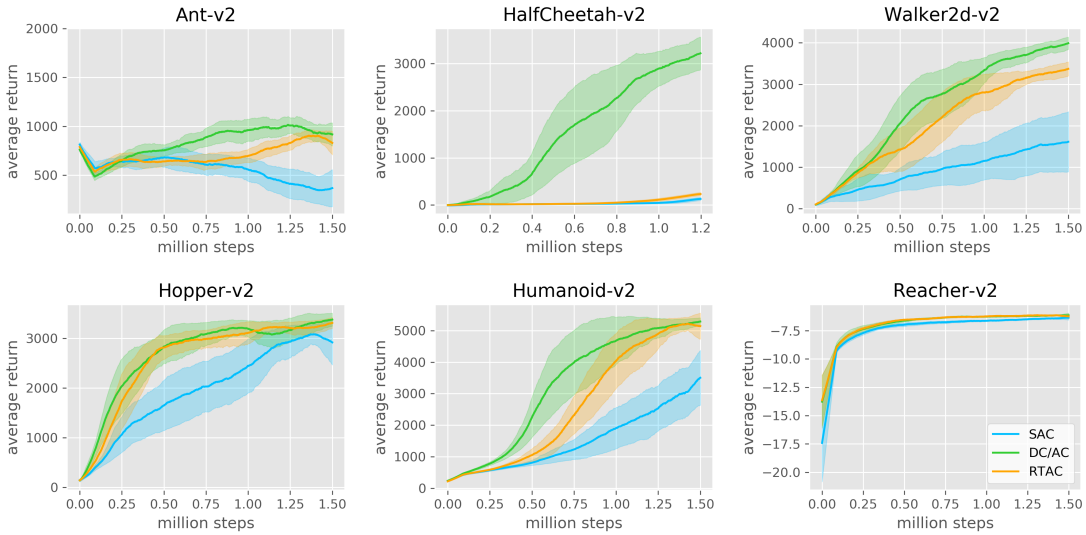

Figure 15: $\omega \in [0;2], \alpha \in [1;3]$ (uniformly sampled delays): This experiment is maybe even more difficult than the WiFi experiment featured in the main paper, because it gives equal probability to all possible delays in the specified ranges (but delays are smaller here which makes it easier for RTAC, because these delays are closer to 1). All tested approaches fail on randomly delayed Ant. For other tasks, the advantage of DCAC is very clear over SAC.

E DEFINITIONS

Definition 8. The n-step state-reward distribution for an environment $E = (S, A, \mu, p)$ and a policy π is defined as

$$p^\pi_{n+1}(\underbrace{s', r', \tau_n}_{\tau_{n+1}}|s) = \mathbb{E}_{a\sim\pi(\cdot|s)}[p^\pi_n(\tau_n|s')p(s', r'|s, a)] = \int_A p^\pi_n(\tau_n|s')p(s', r'|s, a)\pi(a|s)da \quad (13)$$

with the base case $p^\pi_0(s) = 1$ and the first iterate $p^\pi_1(s', r'|s) = \int_A p(s', r'|s, a)\pi(a|s)da$.

Definition 9. A 1-step action-value estimator is defined as

$$\hat{q}_1(s,a;\, s',r') = r' + \gamma\, \mathbb{E}_{a'\sim\pi(\cdot|s')}[\hat{q}_0(s',a')]. \quad (14)$$

Part of this estimator is usually another parametric estimator $\hat{q}_0$ (e.g. a neural network trained with stochastic gradient descent).

F OTHER MATHEMATICAL RESULTS

F.1 LEMMA ON STEADY-STATE VALUE ESTIMATION BIAS

Lemma 3. The expected bias of the n-step value estimator under the steady-state distribution (if it exists) is

$$\mathbb{E}_{x\sim p^\pi_{\text{ss}}}[\text{bias}\, \hat{v}_n(x)] = \gamma^n \mathbb{E}_{x\sim p^\pi_{\text{ss}}}[\text{bias}\, \hat{v}_0(x)] \quad (15)$$

Proof. We remind ourselves that the steady state distribution observes

$$p^\pi_{\text{ss}}(x_n) = \mathbb{E}_{x_0\sim p^\pi_{\text{ss}}}[p^\pi_n(..., x_n, r_n|x_0)]. \quad (16)$$

According to Lemma 1 we then have

$$\mathbb{E}_{x_0\sim p^\pi_{ss}} \operatorname{bias}(\hat{v}_n(x_0,\cdot)) = \gamma^n \mathbb{E}_{\ldots,x^*_n,r^*_n\sim p^\pi_n(\cdot|x_0)}[\operatorname{bias}(\hat{v}_0(x^*_n))] \tag{17}$$

$$= \gamma^n \mathbb{E}_{x\sim p^\pi_{ss}}[\operatorname{bias}\hat{v}_0(x)]. \tag{18}$$

□

F.2 Lemma on a Dirac delta product distribution

Lemma 4. For $p(u,v) = \delta(u-c)q(u,v)$ if $q(u,v) < \infty$ for $u = c$ then $p(u,v) = \delta(u-c)q(c,v)$.

Proof. If $u = c$ then $p(u,v) = \delta(u-c)q(c,v)$, otherwise $p(u,v) = 0 = \delta(u-c)q(c,v)$ □

F.3 Lemma on f

Lemma 5. The dynamics described by f depend neither on the input action nor on a range of actions in the action buffer:

$$f_\Delta(s^*_1, \alpha^*_1, r^*_1 | x_0, a^\mu_0) = f_\Delta(s^*_1, \alpha^*_1, r^*_1 | x^*_0, a^\pi_0)$$

with $x_0 = s_0, u_0, \omega_0, \alpha_0$ and $x^*_0 = s^*_0, u^*_0, \omega^*_0, \alpha^*_0$, given that $s_0, \omega_0, \alpha_0 = s^*_0, \omega^*_0, \alpha^*_0$ and given

$$u_0[\omega^*_0 - \delta + \alpha^*_1] = u^*_0[\omega^*_0 - \delta + \alpha^*_1] \quad \text{for all} \quad \delta \in \{\Delta, \Delta-1, \ldots, 0\}$$

Proof. We prove by induction.

The base case ($\omega^*_0 - \omega^*_1 = -1$) is trivial since it does not depend on the inputs that differ.

For the induction step we have

$$f_\Delta(s^*_1, \alpha^*_1, r^*_1 | s_0, u_0, \omega_0, \alpha_0, a^\mu_0) = \mathbb{E}_{\bar{s},\bar{\alpha},\bar{r}\sim f_{\Delta-1}(\cdot|s_0,u_0,\omega_0,\alpha_0,a^\mu_0)}[p(s^*_1, r^*_1 - \bar{r}|\bar{s}, u[\omega_0 - \Delta + \alpha^*_1])\, p_\alpha(\alpha^*_1|\bar{\alpha})] \tag{19}$$

Because of our condition on u_0 and u^*_0 and the fact that $\omega_0 = \omega^*_0$ this is equal to

$$\mathbb{E}_{\bar{s},\bar{\alpha},\bar{r}\sim f_{\Delta-1}(\cdot|s_0,u_0,\omega_0,\alpha_0,a^\mu_0)}[p(s^*_1, r^*_1 - \bar{r}|\bar{s}, u^*_0[\omega^*_0 - \Delta + \alpha^*_1])\, p_\alpha(\alpha^*_1|\bar{\alpha})]$$

We can now use the induction hypothesis since the conditions on $s_0, u_0, \omega_0, \alpha_0$ are still met when $\Delta \leftarrow \Delta - 1$.

$$\mathbb{E}_{\bar{s},\bar{\alpha},\bar{r}\sim f_{\Delta-1}(\cdot|s^*_0,u^*_0,\omega^*_0,\alpha^*_0,a^\pi_0)}[p(s^*_1, r^*_1 - \bar{r}|\bar{s}, u^*_0[\omega^*_0 - \Delta + \alpha^*_1])\, p_\alpha(\alpha^*_1|\bar{\alpha})] = f_\Delta(s^*_1, \alpha^*_1, r^*_1 | x^*_0, a^\pi_0) \tag{20}$$

□

F.4 LEMMA ON PARTIAL RESAMPLING

Lemma 6. Partially resampling trajectories collected under a policy μ according to σ_n^π transforms them into trajectories distributed according to π.

$$\mathbb{E}_{\tau_n \sim p_n^\mu(\cdot|x_0)}[\sigma_n^\pi(\tau_n^*|x_0^*;\tau_n)] = p_n^\pi(\tau_n^*|x_0^*)$$

with $x_0 = s_0, u_0, \omega_0, \alpha_0$ and $x_0^* = s_0^*, u_0^*, \omega_0^*, \alpha_0^*$, on the condition that $s_0, \omega_0, \alpha_0 = s_0^*, \omega_0^*, \alpha_0^*$ and on the condition that the actions in the initial action buffers u_0 and u_0^* that are applied in the following trajectory are the same, i.e.

$$u_0[k:\text{end}] = u_0^*[k:\text{end}] \quad \text{with} \quad k = \min_i(\omega_{i+1}^* + \alpha_{i+1}^* - i) \quad \text{for} \quad i \in \{0, n-1\}$$

and for the trajectory $\tau_n^* = (s_1^*, u_1^*, \omega_1^*, \alpha_1^*, \ldots, s_n^*, u_n^*, \omega_n^*, \alpha_n^*)$.

Proof. We start with the induction base for $n = 0$. The theorem is trivial in this case since we have 0-length trajectories $()$ and $p_0^\mu(()|x_0) = \sigma_0^\pi(()|x_0^*;()) = p_0^\pi(()|x_0^*) = 1$.

For the induction step we start with the left hand side of the lemma's main equation.

$$\begin{aligned}&\mathbb{E}_{\tau_n \sim p_n^\mu(\cdot|x_0)}[\sigma_n^\pi(\tau_n^*|x_0^*;\tau_n)] \\ &\qquad = \mathbb{E}_{a_0^\mu \sim \mu(\cdot|x_0)}[\mathbb{E}_{x_1,r_1 \sim \tilde{p}(x_1,r_1|x_0,a_0^\mu)}[\mathbb{E}_{\tau_{n-1} \sim p_{n-1}^\mu(\cdot|x_1)}[\sigma_n^\pi(\tau_n^*|x_0^*;x_1,r_1,\tau_{n-1})]]] \end{aligned} \tag{21}$$

with

$$\tilde{p}(s_1,u_1,\omega_1,\alpha_1,r_1|s_0,u_0,\omega_0,\alpha_0,a_0^\mu) = f_{\omega_0-\omega_1}(s_1,\alpha_1,r_1|s_0,u_0,\omega_0,\alpha_0,a_0^\mu)\, p_\omega(\omega_1|\omega_0)\, p_u(u_1|u_0,a_0^\mu)$$

Plugging that and solving the integral over u_1 yields

$$\begin{aligned}&= \mathbb{E}_{a_0^\mu \sim \mu(\cdot|x_0)}[\mathbb{E}_{\omega_1 \sim p_\omega(\cdot|\omega_0)}[\mathbb{E}_{s_1,\alpha_1,r_1 \sim f_{\omega_0-\omega_1}(\cdot|s_0,u_0,\omega_0,\alpha_0,a_0^\mu)}[\\ &\quad \mathbb{E}_{\tau_{n-1} \sim p_{n-1}^\mu(\cdot|s_1,(a_0^\mu,u_0[1:-1]),\omega_1,\alpha_1)}[\sigma_n^\pi(\tau_n^*|x_0^*;s_1,(a^\mu,u_0[1:-1]),\omega_1,\alpha_1,r_1,\tau_{n-1})]]]] \end{aligned} \tag{22}$$

Rolling out σ_n^π by one step and integrating out $s_1, \omega_1, \alpha_1, r_1$ yields

$$\begin{aligned}&= \mathbb{E}_{a_0^\mu \sim \mu(\cdot|x_0)}[\mathbb{E}_{\tau_{n-1} \sim p_{n-1}^\mu(\cdot|s_1^*,(a_0^\mu,u_0[1:-1]),\omega_1^*,\alpha_1^*)}[\mathbb{E}_{a_0^\pi \sim \pi(\cdot|x_0^*)}[\delta(u_1^* - (a_0^\pi, u_0^*[1:-1])) \\ &\qquad \sigma_{n-1}^\pi(\tau_{n-1}^*|s_1^*,u_1^*,\omega_1^*,\alpha_1^*;\tau_{n-1}) f_{\omega_0-\omega_1^*}(s_1^*,\alpha_1^*,r_1^*|s_0,u_0,\omega_0,\alpha_0,a_0^\mu)\, p_\omega(\omega_1^*|\omega_0)]]] \end{aligned} \tag{23}$$

Reordering terms and substituting $s_0, \omega_0, \alpha_0 = s_0^*, \omega_0^*, \alpha_0^*$ yields

$$\begin{aligned}&= p_\omega(\omega_1^*|\omega_0^*)\mathbb{E}_{a_0^\pi \sim \pi(\cdot|x_0^*)}[\delta(u_1^* - (a_0^\pi, u_0^*[1:-1])) \\ &\qquad \mathbb{E}_{a_0^\mu \sim \mu(\cdot|x_0)}[f_{\omega_0^*-\omega_1^*}(s_1^*,\alpha_1^*,r_1^*|x_0,a_0^\mu) \\ &\qquad\quad \mathbb{E}_{\tau_{n-1} \sim p_{n-1}^\mu(\cdot|s_1^*,(a^\mu,u_0[1:-1]),\omega_1^*,\alpha_1^*)}[\sigma_{n-1}^\pi(\tau_{n-1}^*|x_1^*;\tau_{n-1})]]] \end{aligned} \tag{24}$$

We can substitute the f term according to Lemma 5 since the condition between x_0 and x_0^* are met. More precisely the condition on u_0 and u_0^* is met because $k \leq \omega_0^* - \Delta + \alpha_1^* = \omega_1^* + \alpha_1^*$. After the substitution we have

$$\begin{aligned}&= p_\omega(\omega_1^*|\omega_0^*)\mathbb{E}_{a_0^\pi \sim \pi(\cdot|x_0^*)}[\delta(u_1^* - (a_0^\pi, u_0^*[1:-1]))\; f_{\omega_0^*-\omega_1^*}(s_1^*,\alpha_1^*,r_1^*|x_0^*,a_0^\pi) \\ &\qquad \mathbb{E}_{a_0^\mu \sim \mu(\cdot|x_0)}[\mathbb{E}_{\tau_{n-1} \sim p_{n-1}^\mu(\cdot|s_1^*,(a^\mu,u_0[1:-1]),\omega_1^*,\alpha_1^*)}[\sigma_{n-1}^\pi(\tau_{n-1}^*|x_1^*;\tau_{n-1})]]] \end{aligned} \tag{25}$$

We can substitute the induction hypothesis in the following form.

$$\mathbb{E}_{\tau_{n-1}\sim p^{\mu}_{n-1}(\cdot|x_1)}[\sigma^{\pi}_{n-1}(\tau^*_{n-1}|x^*_1;\tau_{n-1})] = p^{\pi}_{n-1}(\tau^*_{n-1}|x^*_1)$$

on the condition that

$$u_1[k:\text{end}] = u^*_1[k:\text{end}] \quad \text{with} \quad k = \min_i(\omega^*_{i+2} + \alpha^*_{i+2} - i) \quad \text{for} \quad i \in \{0, n-2\}$$

for the trajectory $\tau^*_{n-1} = (s^*_2, u^*_2, \omega^*_2, \alpha^*_2, \ldots, s^*_n, u^*_n, \omega^*_n, \alpha^*_n)$. To check that this condition is met we observe that $u_1 = (a^{\mu}_0, u_0[1:-1])$ and substitute $u^*_1 = (a^{\pi}_0, u^*_0[1:-1])$ (made possible by Lemma 4) which means that

$$u_0[k-1:\text{end}] = u^*_0[k-1:\text{end}] \quad \text{with} \quad k = \min_i(\omega^*_{i+2} + \alpha^*_{i+2} - i) \quad \text{for} \quad i \in \{0, n-2\}$$

Substituting the induction hypothesis yields

$$= p_{\omega}(\omega^*_1|\omega^*_0)\mathbb{E}_{a^{\pi}_0\sim\pi(\cdot|x^*_0)}[\delta(u^*_1 - (a^{\pi}_0, u_0[1:-1]))\, f_{\omega^*_0-\omega^*_1}(s^*_1, \alpha^*_1, r^*_1|x^*_0, a^{\pi}_0)\, p^{\pi}_{n-1}(\tau^*_{n-1}|x^*_1)] \tag{26}$$

which is

$$\mathbb{E}_{a^{\pi}_0\sim\pi(\cdot|x^*_0)}[p^{\pi}_{n-1}(\tau^*_{n-1}|x^*_1)\, \tilde{p}(x^*_1, r^*_1|x^*_0, a^{\pi}_0)] = p^{\pi}_n(\tau^*_n|x^*_0)$$

□

G PROOFS OF THE RESULTS FROM THE MAIN PAPER

A

Theorem 1. The partial trajectory resampling operator σ^{π}_n (Def. 3) transforms off-policy trajectories into on-policy trajectories

$$\mathbb{E}_{\tau_n\sim p^{\mu}_n(\cdot|x_0)}[\sigma^{\pi}_n(\tau^*_n|x_0;\tau_n)] = p^{\pi}_n(\tau^*_n|x_0) \tag{3}$$

on the condition that none of the delayed observations depend on any of the resampled actions, i.e.

$$\omega^*_t + \alpha^*_t \geq t \tag{4}$$

where t indexes the trajectory $\tau^*_n = (s^*_1, u^*_1, \omega^*_1, \alpha^*_1, r^*_1, \ldots, s^*_n, u^*_n, \omega^*_n, \alpha^*_n, r^*_n)$ from 1 to n.

Proof. The theorem is a special case of Lemma 6 with $x_0 = x^*_0$. This allows us to simplify the condition in the lemma as we show next.

Since $u_0 = u^*_0$ we can allow all $k \geq 1$ which is the minimum allowed index for u. Therefore we must ensure $1 \leq \min_i(\omega^*_i + \alpha^*_i - i)$. Since the $\min$ must be larger than 1 then all arguments must be larger than 1 which means this is equivalent to

$$1 \leq \omega^*_i + \alpha^*_i - i.$$

This can be transformed into

$$\omega^*_i + \alpha^*_i \geq i \quad \text{for} \quad i \in \{0, n-1\} \tag{27}$$

□

Lemma 1. The n-step value estimator has the following bias:

$$\text{bias}(\hat{v}_n(x_0,\cdot)) = \gamma^n \mathbb{E}_{\ldots,x_n^*,r_n^* \sim p_n^\pi(\cdot|x_0)}[\text{bias}(\hat{v}_0(x_n^*))] \tag{6}$$

Proof.

$$\text{bias}(\hat{v}_n(x_0,\cdot)) = \mathbb{E}_{\tau_n^* \sim p_n^\pi(\cdot|x_0)}[\hat{v}_n(x_0,\tau_n^*) - v^\pi(x_0)] \tag{28}$$
$$= \mathbb{E}_{\tau_n^* \sim p_n^\pi(\cdot|x_0)}[r_1^* + \gamma \hat{v}_{n-1}(x_1^*;\tau_{n-1}^*)] - \mathbb{E}_{a_0 \sim \pi(\cdot|x_0)}[\mathbb{E}_{r_1^*,x_1^* \sim \tilde{p}(\cdot|x_0,a_0)}[r_1^* + \gamma v^\pi(x_1^*)]] \tag{29}$$
$$= \mathbb{E}_{\tau_n^* \sim p_n^\pi(\cdot|x_0)}[r_1^* + \gamma \hat{v}_{n-1}(x_1^*;\tau_{n-1}^*) - r_1^* - \gamma v^\pi(x_1^*)] \tag{30}$$
$$= \gamma \mathbb{E}_{\tau_n^* \sim p_n^\pi(\cdot|x_0)}[\hat{v}_{n-1}(x_1^*;\tau_{n-1}^*) - v^\pi(x_1^*)] \tag{31}$$
$$= \ldots \tag{32}$$
$$= \gamma^n \mathbb{E}_{\tau_n^* \sim p_n^\pi(\cdot|x_0)}[\hat{v}_0(x_n^*) - v^\pi(x_n^*)] \tag{33}$$
$$= \gamma^n \mathbb{E}_{\ldots,x_n^*,r_n^* \sim p_n^\pi(\cdot|x_0)}[\text{bias}(\hat{v}_0(x_n^*))] \tag{34}$$

□

B

Lemma 2. In a RDMDP (E, p_ω, p_α) the soft value function is:

$$v^{\text{soft}}(x_0^*) = \mathbb{E}_{a \sim \pi(\cdot|x_0^*)}[\mathbb{E}_{x_1^*,r_1^* \sim \tilde{p}(\cdot|x_0^*,a)}[r_1^* + \gamma v^{\text{soft}}(x_1^*)] - \log \pi(a|x_0^*)] \tag{7}$$

Proof. The soft value function for an environment $(X, A, \bar{\mu}, \bar{p})$ is defined as

$$v^{\text{soft}}(x_0^*) = \mathbb{E}_{a \sim \pi(\cdot|x_0^*)}[q^{\text{soft}}(x_0^*,a) - \log \pi(a|x_0^*)] \tag{35}$$

where

$$q^{\text{soft}}(x_0^*,a) = \mathbb{E}_{x_1^*,r_1^* \sim \bar{p}(x_0^*,a)}[r_1^* + \gamma v^{\text{soft}}(x_1^*)] \tag{36}$$

If $(X, A, \bar{\mu}, \bar{p}) = \text{RDMDP}(E, p_\omega, p_\alpha) = (X, A, \tilde{\mu}, \tilde{p})$ with $E = (S, A, \mu, p)$ this is

$$q^{\text{soft}}(x_0^*,a) = \mathbb{E}_{x_1^*,r_1^* \sim \tilde{p}(\cdot|x_0^*,a)}[r_1^* + \gamma v^{\text{soft}}(x_1^*)] \tag{37}$$

and

$$v^{\text{soft}}(x_0^*) = \mathbb{E}_{a \sim \pi(\cdot|x_0^*)}[\mathbb{E}_{x_1^*,r_1^* \sim \tilde{p}(\cdot|x_0^*)}[r_1^* + \gamma v^{\text{soft}}(x_1^*)] - \log \pi(a|x_0^*)] \tag{38}$$

□

Proposition 1. The DCAC actor loss is a less biased version of the SAC actor loss with

$$\text{bias}(L_\pi^{\text{DCAC}}) = \mathbb{E}_n[\gamma^n] \, \text{bias}(L_\pi^{\text{SAC}}) \tag{11}$$

assuming both are using similarly biased parametric value estimators to compute the loss, i.e.

$$\text{bias}(\hat{v}_0^{\text{soft}}(x)) = \mathbb{E}_{a \sim \pi(\cdot|x)}[\text{bias}(\hat{q}_0^{\text{soft}}(x,a))] \tag{12}$$

Proof. Note that for simplicity, we also assume that the states in the replay memory are distributed according to the steady-state distribution, i.e. $D \sim p_{\text{ss}}^\pi$. This assumption could be avoided by making more complicated assumptions about the biases of the state-value and action-value estimators.

We now start with the bias of the DCAC loss w.r.t to an unbiased SAC loss using the true action-value function,

$$\text{bias}(L_\pi^{\text{DCAC}}) = L_\pi^{\text{DCAC}} - L_\pi^{\text{SAC -UB}} \tag{39}$$

where

$$L_\pi^{\text{DCAC}} = -\mathbb{E}_{x_0,\tau_n \sim \mathcal{D}} \, \mathbb{E}_{\tau_n^* \sim \sigma_n^\pi(\cdot|x_0;\tau_n)}[\hat{v}_n^{\text{soft}}(x_0;\ \tau_n^*)] \tag{40}$$
$$= -\mathbb{E}_{x_0 \sim D} \mathbb{E}_n \mathbb{E}_{\tau_n^* \sim p_n^\pi(\cdot|x_0)}[\hat{v}_n^{\text{soft}}(x_0;\ \tau_n^*)] \quad | \text{ Theorem 1} \tag{41}$$

and

$$L_\pi^{\text{SAC -UB}} = \mathbb{E}_{x_0 \sim \mathcal{D}}[\mathbb{E}_{a \sim \pi(\cdot|x_0)}[\log \pi(a|x_0) - q^{\text{soft}}(x_0,a)]] \tag{42}$$
$$= \mathbb{E}_{x_0 \sim \mathcal{D}}[v^{\text{soft}}(x_0)]. \tag{43}$$

Substituting these we have

$$
\begin{aligned}
\mathrm{bias}(L_\pi^{\text{DCAC}}) =& \mathbb{E}_{x_0\sim D}\mathbb{E}_n[\hat{v}_n^{\text{soft}}(x_0;\ \tau_n^*) - v^{\text{soft}}(x_0)] && (44)\\
=& \mathbb{E}_{x_0\sim D}\mathbb{E}_n[\mathrm{bias}(\hat{v}_n^{\text{soft}}(x_0;\cdot)] && (45)\\
=& \mathbb{E}_{x_0\sim D}\mathbb{E}_n[\gamma^n \mathbb{E}_{\ldots,x_n,r_n\sim p_n^\pi(\cdot|x_0)}[\mathrm{bias}(\hat{v}_0^{\text{soft}}(x_n))]] \quad | \text{ Lemma 1} && (46)\\
=& \mathbb{E}_n[\gamma^n]\ \mathbb{E}_{x\sim D}[\mathrm{bias}(\hat{v}_0^{\text{soft}}(x))] \quad | \text{ using } D\sim p_{\text{ss}}^\pi \text{ and Lemma 3} && (47)\\
=& \mathbb{E}_n[\gamma^n]\ \mathbb{E}_{x\sim D}[\mathbb{E}_{a\sim\pi(\cdot|x)}[\mathrm{bias}(\hat{q}_0^{\text{soft}}(x,a))]] \quad | \text{ Equation 12} && (48)\\
=& \mathbb{E}_n[\gamma^n]\ \mathbb{E}_{x\sim D}[\mathbb{E}_{a\sim\pi(\cdot|x)}[\hat{q}_0^{\text{soft}}(x,a) - q^{\text{soft}}(x,a)]] && (49)\\
=& \mathbb{E}_n[\gamma^n]\ (L_\pi^{\text{SAC}} - L_\pi^{\text{SAC -UB}}) && (50)\\
=& \mathbb{E}_n[\gamma^n]\ \mathrm{bias}(L_\pi^{\text{SAC}}) && (51)
\end{aligned}
$$

□