Safe and Efficient Robot Learning by Biasing Exploration Towards Expert Demonstrations

Albert Wilcox

May 12, 2023

Acknowledgement

The research presented in this thesis would not have been possible without the collaboration, mentorship and friendliness from those I've had the pleasure to work with, including Professor Ken Goldberg, Ashwin Balakrishna, Nathan Lambert, Brijen Thanajeyan, Justin Kerr, Raven Huang, Ryan Hoque, Alejandro Escontrela, Allie Gu, Max Fu, Chung Min Kim, Rishi Parikh, Vainavi Viswanath, Kaushik Shivakumar, Satvik Sharma, Simeon Adebola, Varun Kamat, and countless others.

Safe and Efficient Robot Learning by Biasing Exploration Towards Expert Demonstrations

Albert Wilcox

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee

Professor Ken Goldberg
Research Advisor

10 May 2023
(Date)

⋆ ⋆ ⋆ ⋆ ⋆ ⋆ ⋆

PAbbeel
Professor Pieter Abbeel
Second Reader

11 May 2023
(Date)

Safe and Efficient Robot Learning by Biasing Exploration Towards Expert Demonstrations

by

Albert Wilcox III

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ken Goldberg, Chair
Professor Pieter Abbeel

Spring 2023

The dissertation of Albert Wilcox III, titled Safe and Efficient Robot Learning by Biasing Exploration Towards Expert Demonstrations, is approved:

Chair ____ Date ______________

____ Date ______________

____ Date ______________

University of California, Berkeley

Safe and Efficient Robot Learning by Biasing Exploration Towards Expert Demonstrations

Abstract

Safe and Efficient Robot Learning by Biasing Exploration Towards Expert Demonstrations

by

Albert Wilcox III

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Ken Goldberg, Chair

Reinforcement learning (RL) has shown impressive results as a framework for learning to complete complex tasks in a wide variety of low- and high-dimensional environments. However, numerous challenges prevent it from being broadly useful as a tool for robotic control. For one, while others have had success training RL algorithms on densely defined reward functions, these can be exceedingly difficult to define and may lead to unintended behaviors. An alternative approach is to learn based on sparse reward functions, using demonstrations to address the additional exploration challenges these sparse reward functions introduce. A second challenge is that while robots running RL algorithms randomly explore in the real world they are known to exhibit dangerous behaviors, damaging themselves, their environments, or even harming people. In this dissertation, we propose the use of offline demonstrations of desirable behavior as a means to guide online exploration, and present two projects using this concept towards addressing problems with efficiency and safety in RL.

A promising strategy for learning in dynamically uncertain environments is requiring that the agent can robustly return to learned safe sets, where task success (and therefore safety) can be guaranteed. While this approach has been successful in low-dimensions, enforcing this constraint in environments with visual observations is exceedingly challenging. We present a novel continuous representation for safe sets by framing it as *a binary classification problem* in a learned latent space, which flexibly scales to image observations. We then present a new algorithm, Latent Space Safe Sets (LS^3), which uses this representation for long-horizon tasks with sparse rewards.

While prior work has used expert demonstrations to improve RL, these algorithms introduce algorithmic complexity and additional hyperparameters, making them hard to implement and tune. We introduce Monte Carlo augmented Actor-Critic (MCAC), a parameter free modification to standard actor-critic algorithms which initializes the replay buffer with demonstrations and computes a modified Q-value by taking the maximum of the standard temporal distance (TD) target and a Monte Carlo estimate of the reward-to-go. This

encourages exploration in the neighborhood of high-performing trajectories by encouraging high Q-values in corresponding regions of the state space.

To my parents

Everything I've been able to do has been because of you. Thank you.

Contents

List of Figures

List of Tables

Acknowledgments

The research presented in this thesis would not have been possible without the collaboration, mentorship and friendliness from numerous people I've had the pleasure to work with.

First and foremost, Professor Ken Goldberg has been an incredible mentor, and working in his group has opened a world of possibilities to me. I thank you for taking the time to help all of the many students in your group and am extremely grateful for having been one of them.

Next, I'd like to extend an ocean of gratitude to Ashwin Balakrishna. Ashwin was one of my first mentors in the lab and came to be a very close friend. He did an incredible job spending time teaching me to do robotics research, and his sense of humor made long days and nights working on robotics experiments bearable. He's also become a trusted source of advice whenever I'm unsure about life decisions. I can't thank Ashwin enough.

I'd also like to extend thanks to other students I've worked directly with on projects both included and not included in this thesis. I collaborated with Brijen Thanajeyan, also one of my first mentors in the lab, on two papers and am grateful for his help shaping my understanding of robotics research. I've also collaborated closely with Justin Kerr multiple times, and will forever be impressed by his skills getting robots to work correctly. Raven Huang, Ryan Hoque and Alejandro Escontrela, Allie Gu have also been excellent collaborators who I'm grateful to have worked with.

Lastly, I'd like to thank those who I never worked with directly but whose friendship and insights have been crucial to my development as a researcher. These people include Max Fu, Chung Min Kim, Rishi Parikh, Vainavi Viswanath, Kaushik Shivakumar, Satvik Sharma, Simeon Adebola, Varun Kamat, Edith Llontop, Amber Xie, Philippe Hansen-Estruch, and countless others.

Chapter 1

Introduction

Reinforcement learning has been successful in learning complex skills in many environments [39, 64, 59], but there are numerous factors making it difficult to apply RL to a wide range of robotic applications. For one, many successful applications have relied on engineers providing dense, informative reward functions, but this is often very challenging [85, 31, 84]. This is particularly problematic for high-dimensional control tasks, in which there may be a large number of factors that influence the agent's objective. In many settings, it may be much easier to provide sparse reward signals that simply convey high-level information about task progress, such as whether an agent has completed a task or has violated a constraint. However, optimizing RL policies given such reward signals can be exceedingly challenging, as sparse reward functions may not be able to meaningfully distinguish between a wide range of different policies. For tasks that take many timesteps to complete, it is often impossible for a randomly exploring agent to ever stumble upon a successful trajectory, making naive RL impossible.

Another key issue preventing the application of RL to a wide range of robotic tasks is safety. Reinforcement learning relies on the agent randomly exploring a wide range of states, and when the agent is a physical robot in the real world, this random exploration can be dangerous. In the worst case, the robot can accidentally damage itself, its environment, or even harm people. While there are a variety of plausible ways to avoid this, one direction we focus on in this dissertation is safe RL, an RL formulation where the agent is subjected to constraints on the probability of constraint violation [67, 73, 74, 81].

The unifying theme behind these two problems is that they both arise due to RL's need for a body of data with informative rewards in order to learn good policies. Sparse reward RL is difficult because it is difficult to find such informative data through random exploration, and safe RL is difficult because while looking for that data, the robot may exhibit dangerous behaviors. To that end, in this dissertation we propose to use offline demonstrations of desired behavior to guide RL exploration towards promising and safe states, presenting two projects that work towards this idea.

For the first project presented here, discussed in further detail in Chapter 3, we developed a model-based RL algorithm seeking to explicitly constrain the agent to stay near familiar states by explicitly constraining latent planning. This work builds on prior methods which efficiently learn safe control policies by learning a 'safe set' of familiar states and using it to guide exploration [51, 50, 72, 74]. While these works do well enabling safe learning in low dimensional settings with simple tasks, they fall short on tasks that require larger datasets or those with higher dimensional observation spaces, as discussed further in Chapter 3. To that end, we introduce a novel algorithm, Latent Space Safe Sets (LS^3), which learns a latent representation space approximates these algorithms within this latent space. We additionally present simulated and physical robotics experiments demonstrating LS^3's ability to learn high-performing policies while minimizing the probability of constraint violation.

For the second project presented here, we focus specifically on the problem of learning to complete sparse reward tasks using offline data, as discussed in Chapter 4. While prior work has used demonstrations to address this setting [49, 46, 74, 73, 81, 42, 18, 12], these algorithms add significant complexity and hyperparameters, making them difficult to implement and tune for different tasks. To that end, we introduce Monte Carlo augmented Actor-Critic, an easy-to-implement and highly effective change that can be made to a wide variety of actor-critic algorithms without the addition of any hyperparameters. At a high level, MCAC works by slightly modifying the Q target values during critic updates to encourage optimism in regions of the state space where replay buffer data has been successful in the past. This synergizes with the offline data of desired behavior to bias the agent towards high-performing demonstrator trajectories, helping the agent to discover high-performing policies. In addition to the algorithm itself, we provide a suite of experiments studying MCAC's effect on a variety of RL algorithms and studying its sensitivity to a variety of RL training variables.

Chapter 2

Related Work

2.1 Safe, Iterative Learning Control

In iterative learning control (ILC), the agent tracks a reference trajectory and uses data from controller rollouts to refine tracking performance [4]. [53, 52, 51] present a new class of algorithms, known as Learning Model Predictive Control (LMPC), which are reference-free and instead iteratively *improve* upon the performance of an initial feasible trajectory. To achieve this, [53, 52, 51] use data from controller rollouts to learn a safe set and value function, with which recursive feasibility, stability, and local optimality can be guaranteed given a known, deterministic nonlinear system or stochastic linear system under certain regularity assumptions. However, a core challenge with these algorithms is that they assume known system dynamics, and cannot be applied to high-dimensional control problems. [74] extends the LMPC framework to higher dimensional settings in which system dynamics are unknown and must be learned, but the visuomotor control setting introduces a number of new challenges as learned system dynamics, safe sets, and value functions must flexibly scale to visual inputs. [50] designs expressive safe sets for fixed policies using neural network classifiers with Lyapunov constraints. In contrast, LS^3 constructs a safe set for an improving policy by optimizing a task cost function instead of uniformly expanding across the state space.

2.2 Model Based Reinforcement Learning

There has been significant recent progress in algorithms which combine ideas from model-based planning and control with deep learning [8, 35, 10, 38, 7, 40]. These algorithms are gaining popularity in the robotics community as they enable leaning complex policies from data while maintaining some of the sample efficiency and safety benefits of classical model-based control techniques. However, these algorithms typically require hand-engineered dense cost functions for task specification, which can often be difficult to provide, especially in high-dimensional spaces. This motivates leveraging demonstrations (possibly suboptimal) to provide an initial signal regarding desirable agent behavior. There has been some prior work on leveraging demonstrations in model-based algorithms such as [48] and [20], which use model-based control with known dynamics to refine initially suboptimal motion plans, and [10], which uses demonstrations to seed a learned dynamics model for fast online adaptation using iLQR [10]. [74, 89] present ILC algorithms which rapidly improve upon suboptimal demonstrations when system dynamics are unknown. However, these algorithms either require knowledge of system dynamics [48, 20] or are limited to low-dimensional state spaces [10, 74, 89] and cannot be flexibly applied to visuomotor control tasks.

2.3 Reinforcement Learning from Pixels

Reinforcement learning and model-based planning from visual observations is gaining significant recent interest as RGB images provide an easily available observation space for robot learning [9, 37]. Recent work has proposed a number of model-free and model-based algorithms that have seen success in laboratory settings in a number of robotic tasks when learning from visual observations [55, 57, 43, 65, 23, 47, 9, 87, 37]. However, two core issues that prevent application of many RL algorithms in practice, inefficient exploration and safety, are significantly exacerbated when learning from high-dimensional visual observations in which the space of possible behaviors is very large and the features required to determine whether the robot is safe are not readily exposed. There has been significant prior work on addressing inefficiencies in exploration for visuomotor control such as latent space planning [15, 37, 87] and goal-conditioned reinforcement learning [47, 43]. However, safe reinforcement learning for visuomotor tasks has received substantially less attention. [73] and [22] present reinforcement learning algorithms which estimate the likelihood of constraint violations to avoid them [73] or reduce the robot's velocity [22]. Unlike these algorithms, which focus on presenting methods to avoid violating user-specified constraints, LS^3 additionally provides consistent task completion during learning by limiting exploration to the neighborhood of prior task successes. This difference makes LS^3 less susceptible to the challenges of unconstrained exploration present in standard model-free reinforcement learning algorithms.

2.4 Reinforcement Learning from Demonstrations

One standard approach for using demonstrations for RL first uses imitation learning [1] to pre-train a policy, and then fine-tunes this policy with on-policy reinforcement learning algorithms [56, 29, 46, 49]. However, initializing with suboptimal demonstrations can hinder learning and using demonstrations to initialize only a policy is inefficient, since they can also be used for Q-value estimation.

Other approaches leverage demonstrations to explicitly constrain agent exploration. [74] propose a model-based RL approach that uses suboptimal demonstrations to iteratively improve performance by ensuring consistent task completion during learning. Similarly, [**jing2020reinforcement**] also uses suboptimal demonstrations to formulate a soft-constraint on exploration. However, a challenge with these approaches is that they introduce substantial algorithm complexity, making it difficult to tune and utilize these algorithms in practice. For example, while [74] does enable iterative improvement upon suboptimal demonstrations, they require learning a model of system dynamics and a density estimator to capture the support of successful trajectories making it challenging to scale to high-dimensional observations.

Finally, many methods introduce auxiliary losses to incorporate demonstrations into policy updates [27, 12, 24]. Deep Deterministic Policy Gradients from Demonstrations (DDPGfD) [79] maintains all the demonstrations in a separate replay buffer and uses prioritized replay to allow reward information to propagate more efficiently. [42], and [18] use similar approaches, where demonstrations are maintained separately from the standard replay buffer and additional policy losses encourage imitating the behavior in the demonstrations. Meanwhile, RL algorithms such as AWAC [41] pretrain using demonstration data to constrain the distribution of actions selected during online exploration. While these methods often work well in practice, they often increase algorithmic complexity and introduce several additional hyperparameters that are difficult and time consuming to tune. By contrast, MCAC does not increase algorithmic complexity, is parameter-free, and can easily be wrapped around any existing actor-critic algorithm.

2.5 Improving Q-Value Estimates

The core contribution of this work is an easy-to-implement, yet highly effective, method for stabilizing actor-critic methods for sparse reward tasks using demonstrations and an augmented Q-value target. There has been substantial literature investigating learning stability challenges in off-policy deep Q-learning and actor-critic algorithms. See [78] for a more thorough treatment of the learning stability challenges introduced by combining function approximation, bootstrapping, and off-policy learning, as well as prior work focused on mitigating these issues.

One class of approaches focuses on developing new ways to compute target Q-values [30, 82, 77, 58]. [77] computes target Q-values with two Q-networks, using one to select actions and the other to measure the value of selected actions, which helps to prevent the Q-value over-estimation commonly observed in practice in many practical applications of Q-learning. TD3 [11] attempts to address overestimation errors by taking the minimum of two separate Q-value estimates, but this can result in underestimation of the true Q-value target. [33] uses an ensemble of critics to adaptively address estimation errors in Q-value targets, but introduces a number of hyperparameters which must be tuned separately for different tasks. [3] and [45] consider a linear combination of a TD-1 target and Monte Carlo target. [83, 58] and [62] consider a number of different estimators of a policy's value via n-step returns, which compute Q-targets using trajectories with n contiguous transitions followed by a terminal evaluation of the Q-value after n steps. Each of these targets make different bias and variance tradeoffs that can affect learning dynamics.

Similar to our work, [82] explore the idea of taking a maximum over a bootstrapped target Q-value (TD-1 target) and a Monte Carlo estimate of the return-to-go to improve fitted Q-iteration. However, [82] focuses on fully offline Q-learning and only considers simple low-dimensional control tasks using Q-learning with linear value approximation. There are numerous reasons to extend these ideas to online deep RL. First, deep RL algorithms are often unstable, and using the ideas in [82] to improve Q estimates is a promising way to alleviate this, as we empirically verify. Second, while offline learning has many important applications, online RL is far more widely studied, and we believe it is useful to study the effects these ideas have in this setting. To the best of our knowledge, MCAC is the first application of these ideas to online actor-critic algorithms with deep function approximation, and we find that it yields surprising improvements in RL performance on complex high-dimensional continuous control tasks.

Chapter 3

LS^3: Latent Space Safe Sets (LS^3)

One promising strategy for efficiently learning safe control policies is to learn a safe set [51, 50], which captures the set of states from which the agent is known to behave safely, which is often reformulated as the set of states where it has previously completed the task. When used to restrict exploration, this safe set can be used to enable highly efficient and safe learning [51, 72, 74], as exploration is restricted to states in which the agent is confident in task success. However, while these safe sets can give rise to algorithms with a number of appealing theoretical properties such as convergence to a goal set, constraint satisfaction, and iterative improvement [51, 72, 52], using them for controller design for practical problems requires developing continuous approximations at the expense of maintaining theoretical guarantees [74]. This choice of continuous approximation is a key element in determining the applications to which these safe sets can be used for control.

Prior works have presented approaches which collect a discrete safe set of states from previously successful trajectories and represent a continuous relaxation of this set by constructing a convex hull of these states [51] or via kernel density estimation with a tophat kernel function [74]. While these approaches have been successful for control tasks with low-dimensional states, extending them to high-dimensional observations presents two key challenges: (1) *scalability:* these prior methods cannot be efficiently applied when the number of observations in prior successful trajectories is large, as querying safe set inclusion scales linearly with number of samples it contains and (2) *representation capacity:* both of these prior approaches do not scale well to high dimensional observations and are limited in the space of continuous sets that they can efficiently represent. Applying these ideas to visuomotor control is even more challenging, since images do not directly expose details about the system state or dynamics that are typically needed for formal controller analysis [51, 72, 2].

This work makes several contributions. First, we introduce a scalable continuous approximation method which makes it possible to leverage safe sets for visuomotor policy learning. The key idea is to reframe the safe set approximation as a *binary classification problem* in a learned latent space, where the objective is to distinguish states from successful trajectories from those in unsuccessful trajectories. Second, we present , a model-based RL algorithm which encourages the agent to maintain plans back to regions in which it is confident in task completion, even when learning in high dimensional spaces. This constraint makes it possible to define a control strategy to (1) improve safely by encouraging consistent task completion (and therefore avoid unsafe behavior) and (2) learn efficiently since the agent only explores promising states in the immediate neighborhood of those in which it was previously successful. Third, we present simulation experiments on 3 visuomotor control tasks which suggest that can learn to improve upon demonstrations more safely and efficiently than prior algorithms. Fourth, we conduct physical experiments on a vision-based cable routing task which suggest that can learn more efficiently than prior algorithms while consistently completing the task and satisfying constraints during learning.

3.1 Individual Contributions

This chapter is adapted from our paper "LS^3: Latent Space Safe Sets for Long-Horizon Visuomotor Control of Sparse Reward Iterative Tasks" [81], and is joint work with Ashwin Balakrishna, Brijen Thananjeyan, Joseph Gonzalez, and Ken Goldberg.

My contributions to the work in this chapter include developing and implementing the algorithm as well as running all the simulated experiments. I also spent substantial time iterating on and carrying out the physical robotics experiments. Finally, I assisted with writing the paper.

Ashwin Balakrishna and Brijen Thanajeyan provided substantial help scoping and advising the project throughout the algorithm development phase. Both of them were very involved with robotics experiments and writing.

Joseph Gonzalez and Ken Goldberg provided help with guiding the direction of the project and editing drafts of the paper.

3.2 Problem Statement

We consider an agent interacting in a finite horizon goal-conditioned Markov Decision Processes (MDP) which can be described with the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, P(\cdot|\cdot,\cdot), R(\cdot,\cdot), \mu, T)$. $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ maps a state and action to a probability distribution over subsequent states, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, μ is the initial state distribution ($s_0 \sim \mu$), and T is the time horizon. In this work, the agent is only provided with RGB image observations $s_t \in \mathbb{R}_+^{W \times H \times 3} = \mathcal{S}$, where W and H are the image width and height in pixels, respectively. We consider iterative tasks, where the agent must reach a fixed goal set $\mathcal{G} \subseteq \mathcal{S}$ as efficiently as possible and the support of μ is small. While there are a number of possible choices of reward functions that would encourage fast convergence to $\mathcal{G}$, providing shaped reward functions can be exceedingly challenging, especially when learning from high dimensional observations. Thus, as in SAVED, we consider a sparse reward function that only indicates task completion: $R(s, a, s') = 0$ if $s' \in \mathcal{G}$ and -1 otherwise. To incorporate constraints, we augment $\mathcal{M}$ with an extra constraint indicator function $\mathcal{C} : \mathcal{S} \rightarrow \{0, 1\}$ which indicates whether a state satisfies user-specified state-space constraints, such as avoiding known obstacles. This is consistent with the modified CMDP formulation used in [73]. We assume that R and $\mathcal{C}$ can be evaluated on the current state of the system, but may be approximated using prior data for use during planning. We make this assumption because in practice we plan over predicted future states, which may not be predicted at sufficiently high fidelity to expose the necessary information to directly evaluate R and $\mathcal{C}$ during planning.

Given a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, we define its expected total return in $\mathcal{M}$ as $R^\pi = \mathbb{E}_{\pi,\mu,P}\left[\sum_t R(s_t, a_t)\right]$. Furthermore, we define $P^\pi_C(s)$ as the probability of future constraint violation (within time horizon T) under policy π from state s. The objective is to maximize the expected return R^π while maintaining a constraint violation probability lower than $\delta_\mathcal{C}$. This can be written formally as follows:

$$\pi^* =_{\pi \in \Pi} \left\{R^\pi : \mathbb{E}_{s_0 \sim \mu}\left[P^\pi_C(s_0)\right] \leq \delta_\mathcal{C}\right\} \tag{3.2.1}$$

We assume that the agent is provided with an offline dataset $\mathcal{D}$ of transitions in the environment of which some subset $\mathcal{D}_{\text{constraint}} \subsetneq \mathcal{D}$ are constraint violating and some subset $\mathcal{D}_{\text{success}} \subsetneq \mathcal{D}$ appear in successful demonstrations from a suboptimal supervisor. As in [73], $\mathcal{D}_{\text{constraint}}$ contains examples of constraint violating behaviors (for example from prior runs of different policies or collected under human supervision) so that the agent can learn about states which violate user-specified constraints.

3.3 Algorithm Details

We describe how LS^3 uses demonstrations and online interaction to safely learn iteratively improving policies. Section 3.3 describes how we learn a low-dimensional latent representation of image observations to facilitate efficient model-based planning. To enable this planning, we learn a probabilistic forward dynamics model as in [7] in the learned latent space and models to estimate whether plans will likely complete the task (Section 3.3) and to estimate future rewards and constraint violations (Section 3.3) from predicted trajectories. In Section 3.3, we discuss how these components are synthesized in LS^3. Dataset $\mathcal{D}$ is expanded using online rollouts of LS^3 and used to update all latent space models (Sections 3.3 and 3.3) after every K rollouts. See Algorithm 1 and the supplement for further details on training procedures and data collection.

Learning a Latent Space for Planning

Learning compressed representations of images has been a popular approach in vision based control to facilitate efficient algorithms for planning and control which can reason about lower dimensional inputs [15, 87, 44, 66, 21, 37]. To learn such a representation, we train a β-variational autoencoder [19] on states in $\mathcal{D}$ to map states to a probability distribution over a d-dimensional latent space $\mathcal{Z}$. The resulting encoder network $f_{\text{enc}}(z|s)$ is then used

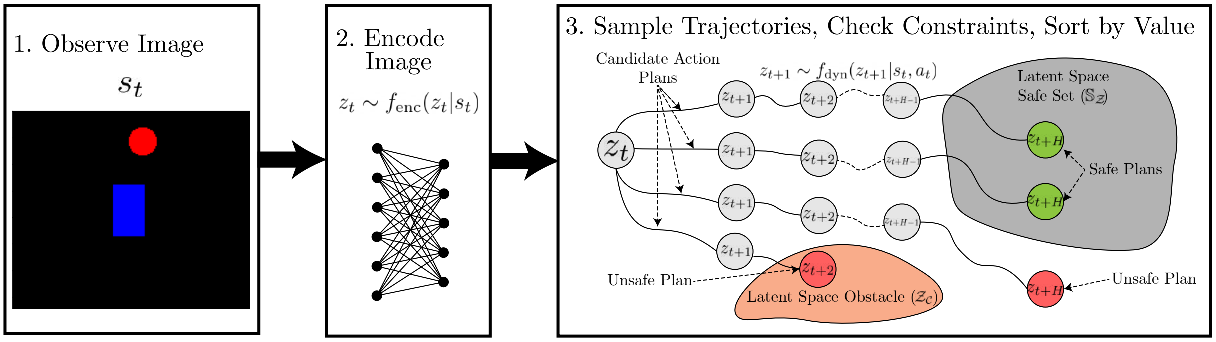

1. Observe Image
s_t
2. Encode Image
$z_t \sim f_{\text{enc}}(z_t|s_t)$
3. Sample Trajectories, Check Constraints, Sort by Value
Candidate Action Plans
$z_{t+1} \sim f_{\text{dyn}}(z_{t+1}|s_t, a_t)$
Latent Space Safe Set ($\mathbb{S}_{\mathcal{Z}}$)
Safe Plans
Unsafe Plan
Latent Space Obstacle ($\mathcal{Z}_{\mathcal{C}}$)
Unsafe Plan
z_t
z_{t+1}
z_{t+2}
z_{t+H-1}
z_{t+H}

Figure 3.1: **Latent Space Safe Sets (LS^3):** At time t, LS^3 observes an image s_t of the environment. The image is first encoded to a latent vector $z_t \sim f_{\text{enc}}(z_t|s_t)$. Then, LS^3 uses a sampling-based optimization procedure to optimize H-length action sequences by sampling H-length latent trajectories over the learned latent dynamics model f_{dyn}. For each sampled trajectory, LS^3 checks whether latent space obstacles are avoided and if the terminal state in the trajectory falls in the latent space safe set. The terminal state constraint encourages the algorithm to maintain plans back to regions of safety and task confidence, but still enables exploration. For feasible trajectories, the sum of rewards and value of the terminal state are computed and used for sorting. LS^3 executes the first action in the optimized plan and then performs this procedure again at the next timestep.

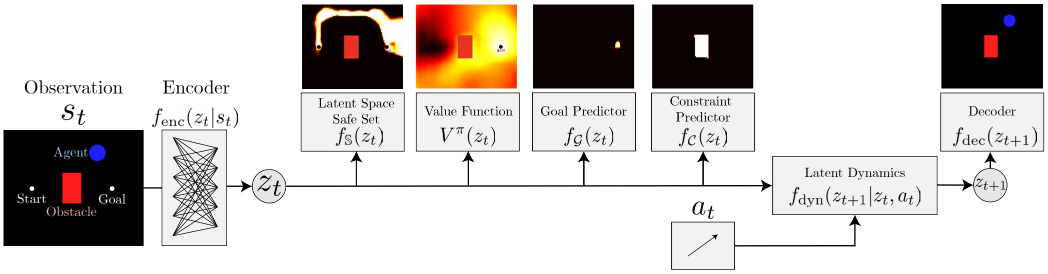


Figure 3.2: **LS^3 Learned Models**: LS^3 learns a low-dimensional latent representation of image-observations (Section 3.3) and learns a dynamics model, value function, reward function, constraint classifier, and safe set for constrained planning and task-completion driven exploration in this learned latent space. These models are then used for model-based planning to maximize the total value of predicted latent states (Section 3.3) while enforcing the safe set (Section 3.3) and user-specified constraints (Section 3.3).

Algorithm 1 Latent Space Safe Sets (LS^3)

```
Require: offline dataset $\mathcal{D}$, number of updates $U$
Train VAE encoder $f_{\text{enc}}$ and decoder $f_{\text{dec}}$ (Section 3.3) using data from $\mathcal{D}$
Train dynamics $f_{\text{dyn}}$, safe set classifier $f_{\mathbb{S}}$(Section 3.3), and the value function $V$ goal
  indicator $f_{\mathcal{G}}$, and constraint estimator $f_{\mathcal{C}}$ (Section 3.3) using data from $\mathcal{D}$.
for $j \in \{1, \ldots, U\}$ do
    for $k \in \{1, \ldots, K\}$ do
        Sample starting state $s_0$ from $\mu$.
        for $t \in \{1, \ldots, T\}$ do
            Choose and execute $a_t$ (Section 3.3)
            Observe $s_{t+1}$, reward $r_t$, constraint $c_t$.
            $\mathcal{D} := \mathcal{D} \cup \{(s_t, a_t, s_{t+1}, r_t, c_t)\}$
        end for
    end for
    Update $f_{\text{dyn}}$, $V$, $f_{\mathcal{G}}$, $f_{\mathcal{C}}$, and $f_{\mathbb{S}}$ with data from $\mathcal{D}$.
end for
```

to sample latent vectors $z_t \sim f_{\text{enc}}(z_t|s_t)$ to train a forward dynamics model, value function, reward estimator, constraint classifier, safe set, and combine these elements to define a policy for model-based planning. Motivated by [34], during training we augment inputs to the encoder with random cropping, which we found to be helpful in learning representations that are useful for planning. For all environments we use a latent dimension of $d = 32$, as in [15] and found that varying d did not significantly affect performance.

We scale all image inputs to a size of $(64, 64, 3)$ before feeding them to the β-VAE, which uses a convolutional neural network for f_{enc} and a transpose convolutional neural network for f_{dec}. We use the encoder and decoder from [15], but modify the second convolutional layer in the encoder to have a stride of 3 rather than 2. As is standard for β-VAEs, we train with a mean-squared error loss combined with a KL-divergence loss. For a particular observation $s_t \in \mathcal{S}$ the loss is

$$J(\theta) = \|f_{\text{dec}}(z_t) - s_t\|_2^2 + \beta D_{KL}\left(f_{\text{enc}}(z_t|s_t)||\mathcal{N}(0,1)\right) \tag{3.3.1}$$

where $z_t \sim f_{\text{enc}}(z_t|s_t)$ is modeled using the reparameterization trick.

Probabilistic Dynamics

As in [5] we train a probabilistic ensemble of neural networks to learn dynamics. Each network has two hidden layers with 128 hidden units. We train these networks with a maximum log-likelihood objective, so for two particular latent states $z_t, z_{t+1} \in \mathcal{Z}$ and the corresponding action $a_t \in \mathcal{A}$ the loss is as follows for dynamics model $f_{\text{dyn},\theta}$ with parameter θ:

$$J(\theta) = -\log f_{\text{dyn},\theta}(z_{t+1}|z_t, a_t) \tag{3.3.2}$$

When using f_{dyn} for planning, we use the TS-1 method from [5].

Latent Safe Sets for Model-Based Control

LS^3 learns a binary classifier for latent states to learn a latent space safe set that represents states from which the agent has high confidence in task completion based on prior experience. Because the agent can reach the goal from these states, they are safe: the agent can avoid constraint violations by simply completing the task as before. While classical algorithms use known dynamics to construct safe sets, we approximate this set using successful trajectories from prior iterations. At each iteration j, the algorithm collects K trajectories in the environment. We then define the sampled safe set at iteration j, $\mathbb{S}^j$, as the set of states from which the agent has successfully navigated to $\mathcal{G}$ in iterations 0 through j of training, where demonstrations trajectories are those collected at iteration 0. We refer to the dataset collecting all these states as $\mathcal{D}_{\text{success}}$. This discrete set is difficult to plan to with continuous-valued state distributions so we leverage data from $\mathcal{D}_{\text{success}}$ (data in the sampled safe set), data from $\mathcal{D} \setminus \mathcal{D}_{\text{success}}$ (data outside the sampled safe set), and the learned encoder from Section 3.3 to learn a continuous relaxation of this set in latent space (the latent safe set). We train a neural network with a binary cross-entropy loss to learn a binary classifier $f_{\mathbb{S}}(\cdot)$ that predicts the probability of a state s_t with encoding z_t being in $\mathbb{S}^j$. To mitigate the negative bias that appears when trajectories that start in safe regions fail, we utilize the intuition that if a state $s_{t+1} \in \mathbb{S}^j$ then it is likely that s_t is also safe. To do this, rather than just predict $\mathbb{1}_{\mathbb{S}^j}$, we train $f_{\mathbb{S}}$ with a recursive objective to predict $\max(\mathbb{1}_{\mathbb{S}^j}, \gamma_{\mathbb{S}} f_{\mathbb{S}}(s_{t+1}))$. The relaxed latent safe set is parameterized by the superlevel sets of $f_{\mathbb{S}}$, where the level $\delta_{\mathbb{S}}$ is adaptively set during execution: $\mathbb{S}^j_{\mathcal{Z}} = \{z_t | f_{\mathbb{S}}(\cdot)(z_t) \geq \delta_{\mathbb{S}}\}$.

The safe set classifier $f_{\mathbb{S}}(\cdot)$ is represented with neural network with 3 hidden layers and 256 hidden units. We train the safe set classifier to predict

$$f_{\mathbb{S}}(s_t) = \max(\mathbb{1}_{\mathbb{S}^j}(s_t), \gamma_{\mathbb{S}} f_{\mathbb{S}}(s_{t+1})) \tag{3.3.3}$$

using a binary cross entropy loss, where $\mathbb{1}_{\mathbb{S}^j}(s_t)$ is an indicator function indicating whether s_t is part of a successful trajectory. Training data is sampled uniformly from a replay buffer containing all of $\mathcal{D}$. Similar to deep value function learning literature [14, 60, 74], the safe set is trained to solve the above equation by fixed point iteration: the safe set is used to construct its own targets, which are then used to update the safe set before using the updated safe set to construct new targets.

Reward and Constraint Estimation

In this work, we define rewards based on whether the agent has reached a state $s \in \mathcal{G}$, but we need rewards that are defined on predictions from the dynamics, which may not correspond to valid real images. To address this, we train a classifier $f_{\mathcal{G}} : \mathcal{Z} \to \{0, 1\}$ to map the encoding of a state to whether the state is contained in $\mathcal{G}$ using terminal states in $\mathcal{D}_{\text{success}}$ (which are known to be in $\mathcal{G}$) and other states in $\mathcal{D}$. However, in the temporally-extended, sparse reward tasks we consider, reward prediction alone is insufficient because rewards only indicate whether the agent is in the goal set, and thus provide no signal on task progress unless the agent can plan to the goal set. To address this, as in prior MPC-literature [74, 72, 51, 75], we train a recursively-defined value function as discussed below. Similar to the reward function, we use the encoder (Section 3.3) to train a classifier $f_{\mathcal{C}} : \mathcal{Z} \rightarrow [0, 1]$ with data of constraint violating states from $\mathcal{D}_{\text{constraint}}$ and the constraint satisfying states in $\mathcal{D} \setminus \mathcal{D}_{\text{constraint}}$ to map the encoding of a state to the probability of constraint violation.

We represent constraint indicator $f_{\mathcal{C}} : \mathcal{Z} \rightarrow \{0, 1\}$ with a neural network with 3 hidden layers and 256 hidden units for each layer with a binary cross entropy loss with transitions from $\mathcal{D}_{\text{constraint}}$ for unsafe examples and the constraint satisfying states in $\mathcal{D} \setminus \mathcal{D}_{\text{constraint}}$ as safe examples. Similarly, we represent the goal estimator $f_{\mathcal{G}} : \mathcal{Z} \rightarrow \{0, 1\}$ with a neural network with 3 hidden layers and 256 hidden units. This estimator is also trained with a binary cross entropy loss with positive examples from $\mathcal{D}_{\text{success}}$ and negative examples sampled from all datasets. For the constraint estimator and goal indicator, training data is sampled uniformly from a replay buffer containing $\mathcal{D}_{\text{success}}$, $\mathcal{D}_{\text{rand}}$ and $\mathcal{D}_{\text{constraint}}$.

We train an ensemble of recursively defined value functions to predict long term reward. We represent these functions using fully connected neural networks with 3 hidden layers with 256 hidden units. Similarly to [74], we use separate training objectives during offline and online training. During offline training, we train the function to predict actual discounted cost-to-go on all trajectories in $\mathcal{D}$. Hence, for a latent vector z_t, the loss during offline training is given as follows where V has parameter θ:

$$J(\theta) = \left(V_\theta^\pi(z_t) - \sum_{i=1}^{T-t} \gamma^i r_{t+i} \right)^2 \tag{3.3.4}$$

In online training we also store target network $V^{\pi'}$ and calculate a temporal difference (TD-1) error,

$$J(\theta) = \Big(V_\theta^\pi(z_t) - (r_t + \gamma V_{\theta'}^{\pi'}(z_{t+1})) \Big)^2 \tag{3.3.5}$$

where θ' are the parameters of a lagged target network and π' is the policy at the timestep at which θ' was set. We update the target network every 100 updates. In each of these equations, γ is a discount factor (we use $\gamma = 0.99$). Because all episodes end by hitting a time horizon, we found it was beneficial to remove the mask multiplier usually used with TD-1 error losses.

For all simulated experiments we update value functions using only data collected by the suboptimal demonstrator or collected online, ignoring offline data collected with random interactions or offline demonstrations of constraint violating behavior.

Model-Based Planning with LS^3

LS^3 aims to maximize total rewards attained in the environment while limiting constraint violation probability within some threshold $\delta_{\mathcal{C}}$ (equation 3.2.1). We optimize an approximation of this objective over an H-step receding horizon with model-predictive control. Precisely, LS^3 solves the following optimization problem to generate an action to execute at timestep t:

$$
\begin{aligned}
{}_{a_{t:t+H-1}\in\mathcal{A}^H} \quad & \mathbb{E}_{z_{t:t+H}}\left[\sum_{i=1}^{H-1} f_{\mathcal{G}}(z_{t+i}) + V^{\pi}(z_{t+H})\right] && (3.3.6)\\
\text{s.t.} \quad & z_t \sim f_{\text{enc}}(z_t|s_t) && (3.3.7)\\
& z_{k+1} \sim f_{\text{dyn}}(z_{k+1}|z_k, a_k)\ \forall k \in \{t, \dots, t+H-1\} && (3.3.8)\\
& \hat{\mathbb{P}}\left(z_{t+H} \in \mathbb{S}_{\mathcal{Z}}^{j-1}\right) \geq 1 - \delta_{\mathbb{S}} && (3.3.9)\\
& \hat{\mathbb{P}}(z_{t+i} \in \mathcal{Z}_{\mathcal{C}}) \leq \delta_{\mathcal{C}}\ \forall i \in \{0, \dots, H-1\} && (3.3.10)
\end{aligned}
$$

In this problem, the expectations and probabilities are taken with respect to the learned, probabilistic dynamics model $f_{\text{dyn}}(z_{t+1}|z_t, a_t)$. The optimization problem is solved approximately using the cross-entropy method (CEM) [54] which is a popular optimizer in model-based RL [5, 74, 72, 86, 73].

The objective function is the expected sum of future rewards if the agent executes $a_{t:t+H-1}$ and then subsequently executes π (equation 3.3.6). First, the current state s_t is encoded to z_t (equation 3.3.7). Then, for a candidate sequence of actions $a_{t:t+H-1}$, an H-step latent trajectory $\{z_{t+1}, \dots, z_{t+H}\}$ is sampled from the learned dynamics f_{dyn} (equation 3.3.8). LS^3 constrains exploration using two chance constraints: (1) the terminal latent state in the plan must fall in the safe set (equation 3.3.9) and (2) all latent states in the trajectory must satisfy user-specified state-space constraints (equation 3.3.10). $\mathcal{Z}_{\mathcal{C}}$ is the set of all latent states such that the corresponding observation is constraint violating. The optimizer estimates constraint satisfaction probabilities for a candidate action sequence by simulating it repeatedly over f_{dyn}. The first chance constraint ensures the agent maintains the ability to return to safe states where it knows how to do the task within H steps if necessary. Because the agent replans at each timestep, the agent need not return to the safe set: during training, the safe set expands, enabling further exploration. In practice, we set $\delta_{\mathbb{S}}$ for the safe set classifier $f_{\mathcal{S}}$ adaptively as described in the supplement. The second chance constraint encourages constraint violation probability of no more than $\delta_{\mathcal{C}}$. After solving the optimization problem, the agent executes the first action in the plan: $\pi(z_t) = a_t$ where a_t is the first element of $a^*_{t:t+H-1}$, observes a new state, and replans.

We use the cross entropy method to solve the optimization problem in equation 3.3.6. We build on the implementation of the cross entropy method provided in [6], which works by sampling $n_{\text{candidate}}$ action sequences from a diagonal Gaussian distribution, simulating each one n_{particle} times over the learned dynamics, and refitting the parameters of the Gaussian on the n_{elite} trajectories with the highest score under equation 3.3.6 where constraints are implemented by assigning large negative rewards to trajectories which violate either the safe

Table 3.1: Hyperparameters for LS^3

Parameter	Navigation	Reacher	Sequential Pushing	Cable Routing
$\delta_{\mathbb{S}}$	0.8	0.5	0.8	0.8
$\delta_{\mathcal{C}}$	0.2	0.2	0.2	0.2
β	1×10^{-6}	1×10^{-6}	1×10^{-6}	1×10^{-6}
H	5	3	3	5
n_{particle}	20	20	20	20
$n_{\text{candidate}}$	1000	1000	1000	2000
n_{elite}	100	100	100	200
$n_{\text{cem\_iters}}$	5	5	5	5
d	32	32	32	32
p_{random}	1.0	1.0	1.0	0.3
Frame Stacking	No	Yes	No	No
Batch Size	256	256	256	256
γ	0.99	0.99	0.99	0.99
$\gamma_{\mathbb{S}}$	0.3	0.3	0.9	0.9

set constraint or user-specified constraints. This process is repeated for $n_{\text{cem\_iters}}$ to iteratively refine the set of sampled trajectories to optimize equation 3.3.6. To improve the optimizer's efficiency on tasks where subsequent actions are often correlated, we sample a proportion $(1 - p_{\text{random}})$ of the optimizer's candidates at the first iteration from the distribution it learned when planning the last action. To avoid local minima, we sample a proportion p_{random} uniformly from the action space. See [5] for more details on the cross entropy method as applied to planning over neural network dynamics models.

As mentioned in Section 3.3, we set $\delta_{\mathbb{S}}$ for the safe set classifier $f_{\mathcal{S}}$ adaptively by checking whether there exists at least one plan which satisfies the safe set constraint at each CEM iteration. If no such plan exists, we multiply $\delta_{\mathbb{S}}$ by 0.8 and re-initialize the optimizer at the first CEM iteration with the new value of $\delta_{\mathbb{S}}$. We initialize $\delta_{\mathbb{S}} = 0.8$.

Hyperparameters

In Table 3.1, we present the hyperparameters used to train and run LS^3. We present details for the constraint thresholds $\delta_{\mathcal{C}}$ and $\delta_{\mathbb{S}}$. We also present the planning horizon H and VAE KL regularization weight β. We present the number of particles sampled over the probabilistic latent dynamics model for a fixed action sequence $n_{\text{particles}}$, which is used to provide an estimated probability of constraint satisfaction and expected rewards. For the cross entropy method, we sample $n_{\text{candidate}}$ action sequences at each iteration, take the best n_{elite} action sequences and then refit the sampling distribution. This process iterates $n_{\text{cem\_iters}}$ times. We also report the latent space dimension d, whether frame stacking is used as input, training batch size, and discount factor γ. Finally, we present values of the safe set bellman coefficient

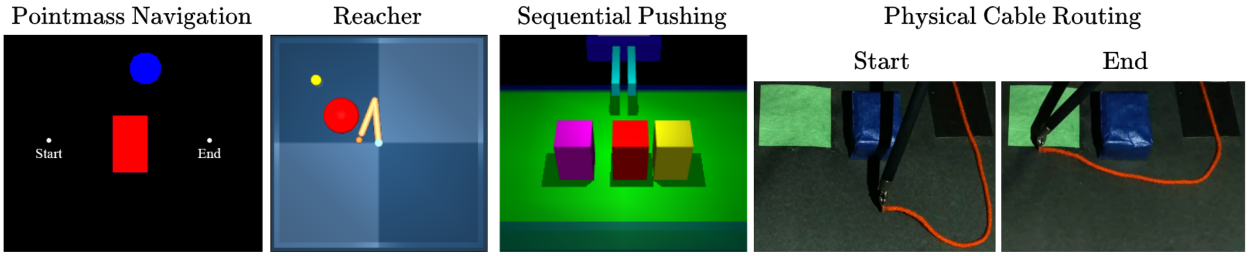


Figure 3.3: **Experimental Domains:** LS^3 is evaluated on 3 long-horizon, image-based, simulation environments: a visual navigation domain where the goal is to navigate the blue point mass to the right goal set while avoiding the red obstacle, a 2 degree of freedom reacher arm where the task is to navigate around a red obstacle to reach the yellow goal set, and a sequential pushing task where the robot must push each of 3 blocks forward a target displacement from left to right. We also evaluate LS^3 on a physical, cable-routing task on a da Vinci Surgical Robot, where the goal is to guide a red cable to a green target without the cable or robot arm colliding with the blue obstacle. This requires learning visual dynamics, because the agent must model how the rest of the cable will deform during manipulation to avoid collisions with the obstacle.

$\gamma_{\mathbb{S}}$. For all domains, we scale RGB observations to a size of $(64, 64, 3)$. For all modules we use the Adam optimizer with a learning rate of 1×10^{-4}, except for dynamics which use a learning rate of 1×10^{-3}.

3.4 Experiments

We evaluate LS^3 on 3 robotic control tasks in simulation and a physical cable routing task on the da Vinci Research Kit (dVRK) [25]. Safe RL is of particular interest for surgical robots such as the dVRK due to its delicate structure, motivating safety, and relatively imprecise controls [74, 61], motivating closed-loop control. We study whether LS^3 can learn more safely and efficiently than algorithms that do not structure exploration based on prior task successes.

Comparisons

We evaluate LS^3 in comparison to prior algorithms that behavior clone suboptimal demonstrations before exploring online (**SACfD**) [13] or leverage offline reinforcement learning to learn a policy using all offline data before updating the policy online (**AWAC**) [41]. For both of these comparisons we enforce constraints via a tuned reward penalty of λ for constraint violations as in [71]. We also implement a version of SACfD with a learned recovery policy (**SACfD+RRL**) using the Recovery RL algorithm [73] to use prior constraint violating data to try to avoid constraint violating states. We then compare LS^3 to an ablated version without the safe set constraint (just binary classification (BC)) in equation 3.3.9 (**LS^3 (−Safe Set)**) to evaluate if the safe set promotes consistent task completion and stable learning.

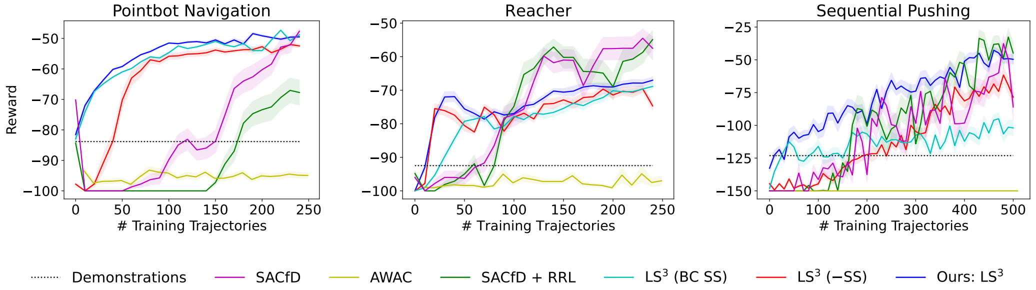


Figure 3.4: **Simulation Experiments Results:** Learning curves showing mean and standard error over 10 random seeds. We see that LS^3 learns more quickly than baselines and ablations. Although SACfD and SACfD+RRL converge to similar reward values, LS^3 is much more sample efficient and stable across random seeds.

Finally, we compare LS^3 to an ablated version of the safe set classifier (Section 3.3) without a recursive objective, where the classifier is just trained to predict $\mathbb{1}_{\mathbb{S}^j}$ (**LS^3 (BC SS)**).

Evaluation Metrics

For each algorithm on each domain, we aggregate statistics over random seeds (10 for simulation experiments, 3 for the physical experiment), reporting the mean and standard error across the seeds. We present learning curves that show the total sum reward for each training trajectory to study how efficiently LS^3 and the comparisons learn each task. Because all tasks use the sparse task completion based rewards defined in Section 3.2, the total reward for a trajectory is the time to reach the goal set, where more negative rewards correspond to slower convergence to $\mathcal{G}$. Thus, for a task with task horizon T, a total reward greater than $-T$ implies successful task completion. The state is frozen in place upon constraint violation until the task horizon elapses. We also report task success and constraint satisfaction rates for LS^3 and comparisons during learning to study (1) the degree to which task completion influences sample efficiency and (2) how safely different algorithms explore. LS^3 collects $K = 10$ trajectories in between training phases on simulated tasks and $K = 5$ in between training phases for physical tasks, while the SACfD and AWAC comparisons update their parameters after each timestep. This presents a metric in terms of the amount of data collected across algorithms.

Domains

In simulation, we evaluate LS^3 on 3 vision-based continuous control domains that are illustrated in Figure 4.1. We evaluate LS^3 and comparisons on a constrained visual navigation task (Pointmass Navigation) where the agent navigates from a fixed start state to a fixed goal set while avoiding a large central obstacle. We study this domain to gain intuition and visualize the learned value function, goal/constraint indicators, and safe set in Figure 3.2.

We then study a constrained image-based reaching task (Reacher) based on [70], where the objective is to navigate the end effector of a 2-link planar robotic arm to a yellow goal position without the end-effector entering a red stay out zone. We then study a challenging sequential image-based robotic pushing domain (Sequential Pushing), in which the objective is to push each of the 3 blocks forward on the table without pushing them to either side and causing them to fall out of the workspace. Finally, we evaluate LS^3 with an image-based physical experiment on the da Vinci Research Kit (dVRK) [26] (Figure 4.1), where the objective is to guide the endpoint of a cable to a goal region without letting the cable or end effector collide with an obstacle. The Pointmass Navigation and Reaching domains have a task horizon of $T = 100$ while the Sequential Pushing domain and physical experiment have task horizons of $T = 150$ and $T = 50$ respectively.

Navigation

The visual navigation domain has 2-D single integrator dynamics with additive Gaussian noise sampled from $\mathcal{N}(0, \sigma^2 I_2)$ where $\sigma = 0.125$. The start position is $(30, 75)$ and goal set is $\mathcal{B}_2((150, 75), 3)$, where $\mathcal{B}_2(c, r)$ is a Euclidean ball centered at c with radius r. The demonstrations are created by guiding the agent north for 20 timesteps, east for 40 timesteps, and directly towards the goal until the episode terminates. This tuned controller ensures that demonstrations avoid the obstacle and also reach the goal set, but they are very suboptimal. To collect demonstrations of constraint violating behavior, we randomly sample starting points throughout the environment, move in a random direction for 15 time steps, and then move directly towards the obstacle. We do not collect additional data for $\mathcal{D}_{\text{rand}}$ in this environment. We collect 50 demonstrations of successful behaviors and 50 trajectories containing constraint violating behaviors.

Reacher

The reacher domain is built on the reacher domain provided in the DeepMind Control Suite from [70]. The robot is represented with a planar 2-link arm and the agent supplies torques to each of the 2 joints. Because velocity is not observable from a single frame, algorithms are provided with several stacked frames as input. The start position of the end-effector is fixed and the objective is to navigate the end effector to a fixed goal set on the top left of the workspace without allowing the end effector to enter a large red stay-out zone. To collect data from $\mathcal{D}_{\text{constraint}}$ we randomly sample starting states in the environment, and then use a PID controller to move towards the constraint. To sample random data that will require the agent to model velocity for accurate prediction, we start trajectories at random places in the environment, and then sample each action from a normal distribution centered around the previous action, $a_{t+1} \sim \mathcal{N}(a_t, \sigma^2 I)$ for $\sigma^2 = 0.2$. We collect 50 demonstrations of successful behavior, 50 trajectories containing constraint violations and 100 short (length 20) trajectories or random data.

Sequential Pushing

This sequential pushing environment is implemented in MuJoCo [76], and the robot can specify a desired planar displacement $a = (\Delta x, \Delta y)$ for the end effector position. The goal is to push all 3 blocks backwards by at least some displacement on the table, but constraints are violated if blocks are pushed backwards off of the table. For the sequential pushing environment, demonstrations are created by guiding the end effector to the center of each block and then moving the end effector in a straight line at a low velocity until the block is in the goal set. This same process is repeated for each of the 3 blocks. Data of constraint violations and random transitions for $\mathcal{D}_{\text{constraint}}$ and $\mathcal{D}_{\text{rand}}$ are collected by randomly switching between a policy that moves towards the blocks and a policy that randomly samples from the action space. We collect 500 demonstrations of successful behavior and 300 trajectories of random and/or constraint violating behavior.

Physical Cable Routing

This task starts with the robot grasping one endpoint of the red cable, and it can make $(\Delta x, \Delta y)$ motions with its end effector. The goal is to guide the red cable to intersect with the green goal set while avoiding the blue obstacle. The ground-truth goal and obstacle checks are performed with color masking. LS^3 and all baselines are provided with a segmentation mask of the cable as input. The demonstrator generates trajectories $\mathcal{D}_{\text{success}}$ by moving the end effector well over the obstacle and to the right before executing a straight line trajectory to the goal set. This ensures that it avoids the obstacle as there is significant margin to the obstacle, but the demonstrations may not be optimal trajectories for the task. Random trajectories $\mathcal{D}_{\text{rand}}$ are collected by following a demonstrator trajectory for some random amount of time and then sampling from the action space until the episode hits the time horizon. We collect 420 demonstrations of successful behavior and 150 random trajectories. We use data augmentation to increase the size of the dataset used to train f_{enc} and f_{dec}, taking the images in $\mathcal{D}$ and creating an expanded dataset by adding randomly sampled affine translations and perspective shifts, until $|\mathcal{D}_{\text{VAE}}| > 100000$.

Simulation Results

We find that LS^3 is able to learn more stably and efficiently than all comparisons across all simulated domains while converging to similar performance within 250 trajectories collected online (Figure 3.4). LS^3 is able to consistently complete the task during learning, while the comparisons, which do not learn a safe set to structure exploration based on prior successes, exhibit much less stable learning. Additionally, in Table 3.2 and Table 3.3, we report the task success rate and constraint violation rate of all algorithms during training. We find that LS^3 achieves a significantly higher task success rate than comparisons on all tasks. We also find that LS^3 violates constraints less often than comparisons on the Reacher task, but violates constraints more often than SACfD and SACfD+RRL on the other domains. This is

Table 3.2: **Task Success Rate over all Training Episodes:** We present the mean and standard error of training-time task completion rate over 10 random seeds. We find LS^3 outperforms all comparisons across all 3 domains, with the gap increasing for the challenging sequential pushing task.

	SACfD	AWAC	SACfD+RRL	LS^3 (−SS)	LS^3
Pointmass Navigation	0.363 ± 0.068	0.312 ± 0.093	0.184 ± 0.053	0.818 ± 0.019	$\mathbf{0.988 \pm 0.004}$
Reacher	0.502 ± 0.072	0.255 ± 0.089	0.473 ± 0.056	0.736 ± 0.025	$\mathbf{0.870 \pm 0.024}$
Sequential Pushing	0.425 ± 0.064	0.006 ± 0.003	0.466 ± 0.065	0.366 ± 0.030	$\mathbf{0.648 \pm 0.049}$

Table 3.3: **Constraint Violation Rate:** We report mean and standard error of training-time constraint violation rate over 10 random seeds. LS^3 violates constraints less than comparisons on the Reacher task, but SAC and SACfD+RRL achieve lower constraint violation rates on the Navigation and Pushing tasks, likely due to spending less time in the neighborhood of constraint violating regions due to their much lower task success rates.

	SACfD	AWAC	SACfD+RRL	LS^3 (−SS)	LS^3
Pointmass Navigation	0.006 ± 0.002	0.104 ± 0.070	$\mathbf{0.001 \pm 0.001}$	0.019 ± 0.006	0.005 ± 0.001
Reacher	0.146 ± 0.039	0.398 ± 0.107	0.142 ± 0.031	0.247 ± 0.027	$\mathbf{0.102 \pm 0.027}$
Sequential Pushing	$\mathbf{0.033 \pm 0.003}$	0.138 ± 0.028	0.054 ± 0.006	0.122 ± 0.031	0.107 ± 0.016

because SACfD and SACfD+RRL spend much less time in the neighborhood of constraint violating states during training due to their lower task success rates. Because they do not efficiently learn to perform the tasks, they do not violate constraints as often. We find that the AWAC comparison achieves very low task performance. While AWAC is designed for offline reinforcement learning, to the best of our knowledge, it has not been previously evaluated on long-horizon, image-based tasks as in this paper, which we hypothesize are very challenging for it.

We find LS^3 has a lower success rate when the safe set constraint is removed (LS^3(−Safe Set)) as expected. The safe set is particularly important in the sequential pushing task, and LS^3 (−Safe Set) has a much lower task completion rate than LS^3. LS^3 without the recursive classification objective from Section 3.3 (LS^3 (BC SS)) has similar performance to LS^3 on the navigation environment, but learns substantially more slowly on the Reacher environment and performs significantly worse than LS^3 on the more challenging Pushing environment as the learned safe set is unable to exploit temporal structure to distinguish safe states from unsafe states. See the supplement for details on experimental parameters and offline data used for LS^3 and comparisons and ablations studying the effect of the planning horizon and threshold used to define the safe set.

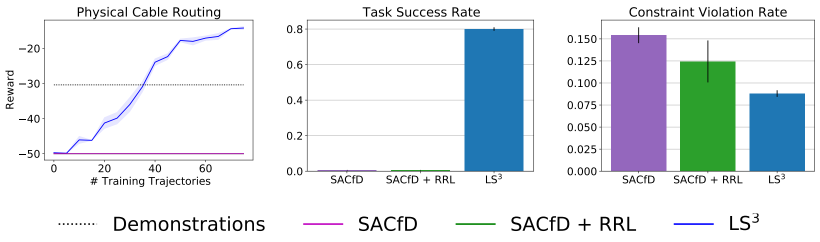


Figure 3.5: **Physical Cable Routing Results:** We present learning curves, task success rates and constraint violation rates with a mean and standard error across 3 random seeds. LS^3 learns a more efficient policy than the demonstrator while still violating constraints less than comparisons, which are unable to learn the task.

Physical Results

In physical experiments, we compare LS^3 to SACfD and SACfD+RRL (Figure 3.5) on the physical cable routing task illustrated in Figure 4.1. We find LS^3 quickly outperforms the suboptimal demonstrations while succeeding at the task significantly more often than both comparisons, which are unable to learn the task and also violate constraints more than LS^3. We hypothesize that the difficulty of reasoning about cable collisions and deformation from images makes it challenging for prior algorithms to make sufficient task progress as they do not use prior successes to structure exploration. See the supplement for details on experimental parameters and offline data used for LS^3 and comparisons.

Additional Results

We additionally study how the task success rate of LS^3 and comparisons evolves as training progresses in Figure 3.6. Precisely, we checkpoint each policy after each training trajectory and evaluate it over 10 rollouts for each of the 10 random seeds (100 total trials per datapoint). We find that LS^3 achieves a much higher task success rate than comparisons early on in training, and maintains a higher task success rate throughout the course of training on all simulation domains.

Sensitivity Experiments

Key hyperparameters in LS^3 are the constraint threshold $\delta_{\mathcal{C}}$ and safe set threshold $\delta_{\mathbb{S}}$, which control whether the agent decides predicted states are constraint violating or in the safe set respectively. We ablate these parameters for the Sequential Pushing environment in Figures 3.7 and 3.9. We find that lower values of $\delta_{\mathcal{C}}$ made the agent less likely to violate constraints as expected. Additionally, we find that higher values of $\delta_{\mathbb{S}}$ helped constrain exploration more effectively, but too high of a threshold led to poor performance suffered as the agent exploited local maxima in the safe set estimation. Finally, we ablate the planning horizon H for LS^3 and find that when H is too high, Latent Space Safe Sets (LS^3) can explore

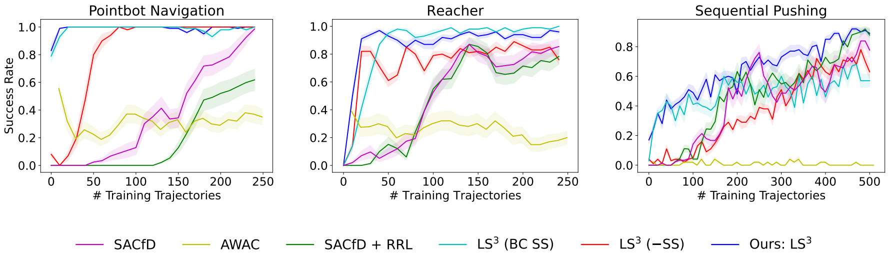


Figure 3.6: **Task Success Rate:** Learning curves showing mean and standard error of task success rate of checkpointed policies over 10 random seeds (and 10 rollouts per seed). We see that LS^3 has a much higher task success rate than comparisons early on, and maintains a success rate at least as high as comparisons throughout training in all environments.

too aggressively away from the safe set, leading to poor performance. When H is lower, LS^3 explores much more stably, but if it is too low (ie. $H = 1$), LS^3 is eventually unable to explore significantly new plans, while slightly increasing H (ie. $H = 3$) allows for continuous improvement in performance.

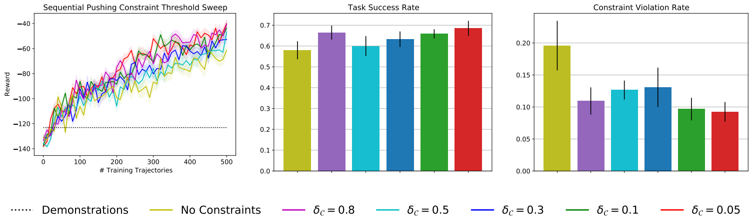


Figure 3.7: **Hyperparameter Sweep for LS^3 Constraint Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathcal{C}}$ on the sequential pushing environment. As expected, we see that without avoiding latent space obstacles (No Constraints) the agent violates constraints more often, while lower thresholds (meaning the planning algorithm is more conservative) generally lead to fewer violations.

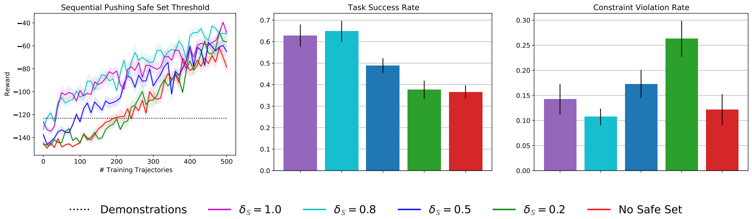


Figure 3.8: **Hyperparameter Sweep for LS^3 Safe Set Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathbb{S}}$ on the sequential pushing environment. We see that after offline training, the agent can successfully complete the task only when $\delta_{\mathbb{S}}$ is high enough to sufficiently guide exploration, and that runs with higher values of $\delta_{\mathbb{S}}$ are more successful overall.

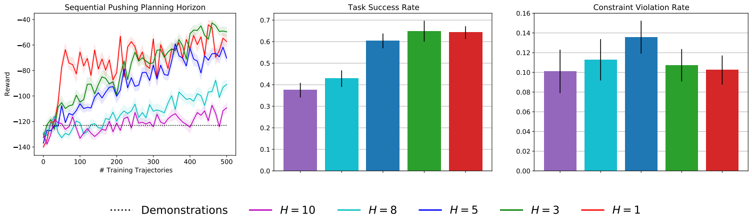


Figure 3.9: **Hyperparameter Sweep for LS^3 Planning Horizon:** Plots show mean and standard error over 10 random seeds for experiments with different settings of H on the sequential pushing environment. We see that when the planning horizon is too high the agent cannot reliably complete the task due to modeling errors. When the planning horizon is too low, it learns quickly but cannot significantly improve because it is constrained to the safe set. We found $H = 3$ to balance this trade off best.

Chapter 4

MCAC: Monte Carlo augmented Actor-Critic

This issue can be mitigated by leveraging demonstrations, which provide initial signal about desired behaviors. Though demonstrations may often be suboptimal in practice, they should still serve to encourage exploration in promising regions of the state space, while allowing the agent to explore behaviors which may outperform the demonstrations. Prior work has considered a number of ways to leverage demonstrations to improve learning efficiency for reinforcement learning, including by initializing the policy to match the behavior of the demonstrator [49, 46], using demonstrations to explicitly constrain agent exploration [74, 73, 81], and introducing auxiliary losses to incorporate demonstration data into policy updates [42, 79, 12]. While these algorithms have shown impressive performance in improving the sample efficiency of RL algorithms, they add significant complexity and hyperparameters, making them difficult to implement and tune for different tasks.

We present Monte Carlo augmented Actor-Critic (MCAC), which introduces an easy-to-implement, but highly effective, change that can be readily applied to existing actor-critic algorithms without the introduction of any additional hyperparameters and only minimal additional complexity. The idea is to encourage initial optimism in the neighborhood of successful trajectories, and progressively reduce this optimism during learning so that it can continue to explore new behaviors. To operationalize this idea, MCAC introduces two modifications to existing actor-critic algorithms. First, MCAC initializes the replay buffer with task demonstrations. Second, MCAC computes a modified target Q-value for critic updates by taking the maximum of the standard temporal distance targets used in existing actor critic algorithms and a Monte Carlo estimate of the reward-to-go. The intuition is that Monte Carlo value estimates can more effectively capture longer-term reward information, making it possible to rapidly propagate reward information from demonstrations through the learned Q-function. This makes it possible to prevent underestimation of values in high-performing trajectories early in learning, as high rewards obtained later in a trajectory may be difficult to initially propagate back to earlier states with purely temporal distance targets [82]. Experiments on five continuous control domains suggest that MCAC is able to substantially accelerate exploration for both standard RL algorithms and recent RL from demonstrations algorithms in sparse reward tasks.

4.1 Individual Contributions

This chapter is adapted from our paper "Monte Carlo Augmented Actor-Critic for Sparse Reward Deep Reinforcement Learning from Suboptimal Demonstrations" [80], and is joint work with Ashwin Balakrishna, Jules Dedieu, Wyame Benslimane, Daniel S. Brown and Ken Goldberg.

I fully led this project, developing it from an idea I discovered while experimenting for a different project to a finished product. Specifically, I implemented the algorithm and ran all experiments in the paper, and did a substantial chunk of the writing work.

Ashwin and Daniel were both very helpful with advising and guiding the project, as well as contributing to the writing. Jules and Wyame helped with implementing Conservative Q Learning.

4.2 Problem Statement

We consider a Markov Decision Process (MDP) described by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, T)$ with a state set $\mathcal{S}$, an action set $\mathcal{A}$, a transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor γ, and finite time horizon T. In each state $s_t \in \mathcal{S}$ the agent chooses an action $a_t \in \mathcal{A}$ and observes the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$ and a reward $r(s_t, a_t) \in \mathbb{R}$. The agent acts according to a policy π, which induces a probability distribution over $\mathcal{A}$ given the state, $\pi(a_t|s_t)$. The agent's goal is to find the policy π^* which at any given $s_t \in \mathcal{S}$ maximizes the expected discounted sum of rewards,

$$\pi^* =_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right], \tag{4.2.1}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots s_T)$ and $\tau \sim \pi$ indicates the distribution of trajectories induced by evaluating policy π in the MDP.

We make additional assumptions specific to the class of problems we study. First, we assume that all transitions in the replay buffer are elements of complete trajectories; this is a reasonable assumption as long as all transitions are collected from rolling out some policy in the MDP. Second, we assume the agent has access to an offline dataset $\mathcal{D}_{\text{offline}}$ of (possibly suboptimal) task demonstrations. Finally, we focus on settings where the reward function is sparse in the sense that most state transitions do not induce a change in reward, causing significant exploration challenges for RL algorithms.

4.3 Preliminaries: Actor-Critic Algorithms

For a given policy π, its state-action value function Q^π is defined as

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k)\right]. \tag{4.3.1}$$

Actor-critic algorithms learn a sample-based approximation to Q^π, denoted Q^π_θ, and a policy π_ϕ which selects actions to maximize Q^π_θ, with a function approximator (typically a neural network) parameterized by θ and ϕ respectively. During the learning process, they alternate between regressing Q^π_θ to predict Q^π and optimizing π_ϕ to select actions with high values under Q^π_θ.

Exactly computing Q^π targets to train Q^π_θ is typically intractable for arbitrary policies in continuous MDPs, motivating other methods for estimating them. One such method is to simply collect trajectories $(s_t, a_t, r_t, s_{t+1}, \ldots s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ by executing the learned policy π_ϕ from state s_t, computing a Monte Carlo estimate of the reward-to-go defined as follows:

$$Q^{\text{target}}_{\text{MC}}(s_t, a_t) = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k), \tag{4.3.2}$$

and fitting Q^π_θ to these targets.

However, $Q^{\text{target}}_{\text{MC}}$ can be a high variance estimator of the reward-to-go [58, 68], motivating the popular one-step temporal difference target (TD-1 target) to help stabilize learning:

$$Q^{\text{target}}_{\text{TD}}(s_t, a_t) = r(s_t, a_t) + \gamma Q^\pi_{\theta'}(s_{t+1}, a_{t+1}), \tag{4.3.3}$$

where $a_{t+1} \sim \pi_\phi(s_{t+1})$, which is recursively defined based on only a single (s_t, a_t, s_{t+1}, r_t) transition. Here θ' is the parameters of a lagged target network as in [77]. There has also been interest in computing TD-n targets, which instead sum rewards for n timesteps and then use Q-values from step $n + 1$ [83, 58, 62].

4.4 Monte Carlo augmented Actor-Critic

MCAC Algorithm

The objective of MCAC is to efficiently convey information about sparse rewards from suboptimal demonstrations to Q^π_θ in order to accelerate policy learning while still maintaining learning stability. To do this, MCAC combines two different methods of computing targets for fitting Q-functions to enable efficient value propagation throughout the state-action space while also learning a Q-value estimator with low enough variance for stable learning. To operationalize this idea, MCAC defines a new Q-function target for training Q^π_θ by taking the maximum of the Monte Carlo Q-target (eq 4.3.2) and the temporal difference Q-target (eq 4.3.3): $\max\left[Q^{\text{target}}_{\text{TD}}(s_t, a_t), Q^{\text{target}}_{\text{MC}}(s_t, a_t)\right]$.

The idea here is that early in learning, a Q-function trained only with temporal difference targets will have very low values throughout the state-action space as it may be very difficult to propagate information about delayed rewards through the temporal difference target for long-horizon tasks with sparse rewards [78]. On the other hand, the Monte Carlo Q-target can easily capture long-term rewards, but often dramatically underestimates Q-values for poorly performing trajectories [82]. Thus, taking a maximum over these two targets serves to initially boost the Q-values for transitions near high performing trajectories while limiting the influence of underestimates from the Monte Carlo estimate.

MCAC can also be viewed as a convenient way to balance the bias and variance properties of the Monte Carlo and temporal difference Q-targets. The Monte Carlo Q-target is well known to be an unbiased estimator of policy return, but have high variance [82]. Conversely, temporal difference targets are known to be biased, but have much lower variance [82, 77]. Thus, as the temporal difference target is typically negatively biased (an underestimate of the true policy return) on successful trajectories early in learning due to the challenge of effective value propagation, computing the maximum of the temporal difference and Monte Carlo targets helps push the MCAC target value closer to the true policy return. Conversely, the Monte Carlo targets are typically pessimistic on unsuccessful trajectories, and their high variance makes it difficult for them to generalize sufficiently to distinguish between unsuccessful trajectories that are close to being successful and those that are not. Thus, computing the maximum of the temporal difference and Monte Carlo targets also helps to prevent excessive pessimism in evaluating unsuccessful trajectories. Notably, MCAC does not constrain its Q-targets explicitly based on the transitions in the demonstrations, making it possible for the policy to discover higher performing behaviors than those in the demonstrations as demonstrated in Section 4.5.

MCAC Practical Implementation

MCAC can be implemented as a wrapper around any actor-critic RL algorithm; we consider 6 options in experiments (Section 4.5). MCAC starts with an offline dataset of suboptimal demonstrations $\mathcal{D}_{\text{offline}}$, which are used to initialize a replay buffer $\mathcal{R}$. Then, during each episode i, we collect a full trajectory τ^i, where the j^{th} transition $(s^i_j, a^i_j, s^i_{j+1}, r^i_j)$ in τ^i is denoted by τ^i_j.

Next, consider any actor-critic method using a learned Q function approximator $Q_\theta(s_t, a_t)$ that, for a given transition $\tau^i_j = (s^i_j, a^i_j, s^i_{j+1}, r^i_j) \in \tau^i \subsetneq \mathcal{R}$ is updated by minimizing the following loss:

$$J(\theta) = \ell\left(Q_\theta(s^i_j, a^i_j), Q^{\text{target}}(s^i_j, a^i_j)\right), \tag{4.4.1}$$

where ℓ is an arbitrary differentiable loss function and Q^{target} is the target value for regressing Q_θ. We note that Q^{target} is defined by the choice of actor-critic method. To implement MCAC, we first calibrate the Monte Carlo targets with temporal difference targets (which provide infinite-horizon Q estimates) by computing the infinite horizon analogue of the Monte Carlo target defined in Equation 4.3.2, which assumes the last observed reward value will

Algorithm 2 Monte Carlo augmented Actor-Critic

```
Require: Offline dataset $\mathcal{D}_{\text{offline}}$.
Require: Total training episodes $N$, batch size $M$, Pretraining Steps $N_p$
Require: Episode time horizon $T$.
Initialize replay buffer $\mathcal{R} := \mathcal{D}_{\text{offline}}$.
Initialize agent $\pi_\phi$ and critic $Q^\pi_\theta$ using data from $\mathcal{D}_{\text{offline}}$.
for $i \in \{1, \dots, N_p\}$ do
    Sample $\mathcal{B} \subsetneq \mathcal{R}$ such that $|\mathcal{B}| = M$.
    Optimize $Q^\pi_\theta$ on $\mathcal{B}$ to minimize loss in eq (4.4.4). Optimize policy $\pi_\phi$ to maximize $Q_\theta$.
end for
for $i \in \{1, \dots, N\}$ do
    Initialize episode buffer $\mathcal{E} = \{\}$.
    Observe state $s^i_1$.
    for $j \in \{1, \dots, T\}$ do
        Sample and execute $a^i_t \sim \pi_\theta(s^i_j)$, observing $s^i_{j+1}, r^i_j$.
        $\tau^i_j \leftarrow (s^i_j, a^i_j, s^i_{j+1}, r^i_j)$
        $\mathcal{E} \leftarrow \mathcal{E} \cup \{\tau^i_j\}$.
        Sample $\mathcal{B} \subsetneq \mathcal{R}$ such that $|\mathcal{B}| = M$.
        Optimize $Q^\pi_\theta$ on $\mathcal{B}$ to minimize loss in eq (4.4.4). Optimize policy $\pi_\phi$ to maximize $Q_\theta$.
    end for
    for $\tau^i_j \in \mathcal{E}$ do
        Compute $Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j)$ as in eq (4.4.2).
        $\tau^i_j \leftarrow (s^i_j, a^i_j, s^i_{j+1}, r^i_j, Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j))$.
    end for
    $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{E}$.
end for
```

repeat forever and uses this to add an infinite sum of discounted rewards, and is given by

$$Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j) = \gamma^{T-j+1}\frac{r^i_T}{1-\gamma} + \sum_{k=j}^{T} \gamma^{k-j} r(s^i_k, a^i_k). \tag{4.4.2}$$

Then, we simply replace the target with a maximum over the original target and the Monte Carlo target defined in Equation 4.4.2, given by

$$Q^{\text{target}}_{\text{MCAC}}(s^i_j, a^i_j) = \max\left[Q^{\text{target}}(s^i_j, a^i_j), Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j)\right]. \tag{4.4.3}$$

This results in the following loss function for training Q_θ:

$$J(\theta) = \ell\left(Q_\theta(s^i_j, a^i_j), Q^{\text{target}}_{\text{MCAC}}(s^i_j, a^i_j)\right). \tag{4.4.4}$$

The full MCAC training procedure (Algorithm 2) alternates between updating Q^π_θ using the method described above, followed by optimizing the policy π_ϕ using any standard policy update method.

4.5 Experiments

In the following experiments we study (1) whether MCAC enables more efficient learning when built on top of standard actor-critic RL algorithms and (2) whether MCAC can be applied to improve prior algorithms for RL from demonstrations. See the supplementary material for code and instructions on how to run experiments for reproducing all results in the paper and additional experiments studying the impact of demonstration quantity, quality, and other algorithmic choices such as whether to pretrain learned networks on demonstration data before online interaction.

Experimental Procedure

All experiments were run on a set of 24 Tesla V100 GPUs through a combination of Google Cloud resources and a dedicated lab server. We aggregate statistics over 10 random seeds for all experiments, reporting the mean and standard error across the seeds with exponential smoothing. Details on hyperparameters and implementation details are provided in the supplementary material.

Domains

We consider the five long-horizon continuous control tasks shown in Figure 4.1. All tasks have relatively sparse rewards, making demonstrations critical for performance. We found that without demonstrations, SAC and TD3 made little to no progress on these tasks.

Pointmass Navigation: The first domain is a pointmass 2D navigation task (Figure 4.1(a)) with time horizon $T = 100$, where the objective is to navigate around the red barrier from start set $\mathcal{S}$ to a goal set $\mathcal{G}$ by executing 2D delta-position controls. If the agent collides with the barrier it receives a reward of -100 and the episode terminates. At each time step, the agent receives a reward of -1 if it is not in the goal set and 0 if it is in the goal set. To increase the difficulty of the task, we perturb the state with zero-mean Gaussian noise at each timestep. The combination of noisy transitions and sparse reward signal makes this a very difficult exploration task where the agent must learn to make it through the slit without converging to the local optima of avoiding both the barrier and the slit.

The demonstrator is implemented as a series of proportional controllers which guide it from the starting set to the slit, through the slit, and to the goal set. The actions are clipped to fit in the action space, and trajectories are nearly optimal. The agent is provided with 20 demonstrations.

Object Manipulation in MuJoCo: We next consider two object manipulation tasks designed in the MuJoCo physics simulator [76], where the objective is to extract a block from a tight configuration on a table (Block Extraction, Figure 4.1(b)) and push each of 3 blocks forward on the plane (Sequential Pushing, Figure 4.1(c)). In the Block Extraction task, the action space consists of 3D delta position controls and an extra action dimension to control the degree to which the gripper is opened. In the Sequential Pushing environment, this extra action dimension is omitted and the gripper is kept closed. In the Block Extraction domain, the agent receives a reward of -1 for every timestep that it hasn't retrieved the red block and 0 when it has. In the Sequential Pushing domain, the reward increases by 1 for each block the agent pushes forward. Thus, the agent receives a reward of -3 when it has made no progress and 0 when it has completed the task. The Block Extraction task is adapted from [73] while the Sequential Pushing task is adapted from [81]. We use a time horizon of $T = 50$ for the Block Extraction task and a longer $T = 150$ for the Sequential Pushing task since it is of greater complexity.

The block extraction demonstrator is implemented as a series of proportional controllers guiding the arm to a position to grip the block, followed by an instruction to close the gripper and a controller to lift. We provide the agent with 50 demonstrations with zero-mean noise injected in the controls to induce suboptimality. For the sequential pushing environment, the demonstrator uses a series of proportional controllers to slowly push one block forward, move backwards, line up with the next block, and repeat the motion until all blocks have been pushed. Because it moves slowly and moves far back from each block it pushes, demonstrations are very suboptimal. For Sequential Pushing, the agent is provided with 500 demonstrations due to the increased difficulty of the task.

Robosuite Object Manipulation: Finally, we consider two object manipulation tasks built on top of Robosuite [88], a collection of robot simulation tasks using the MuJoCo physics engine. We consider the Door Opening task (Figure 4.1(d)) and the Block Lifting task (Figure 4.1(e)). In the Door Opening task, a Panda robot with 7 DoF and a parallel-jaw gripper must turn the handle of a door in order to open it. The door's location is randomized at the start of each episode. The agent receives a reward of -1 if it has not opened the door and a reward of 0 if it has. In the Block Lifting task, the same Panda robot is placed in front of a table with a single block on its surface. The robot must pick up the block and lift it above a certain threshold height. The block's location is randomized for each episode and the agent receives a reward of -1 for every timestep it has not lifted the block and a reward of 0 when it has. Both Robosuite tasks use a time horizon of $T = 50$.

For both Robosuite tasks, demonstrators are trained using SAC on a version of the task with a hand-designed dense reward function, as in the Robosuite benchmarking experiments [88]. In order to ensure suboptimality, we stop training the demonstrator policy before convergence. For each Robosuite environment we use the trained demonstrator policies to generate 100 suboptimal demonstrations for training MCAC and the baselines.

Algorithm Comparisons

We empirically evaluate the following baselines both individually and in combination with MCAC. All methods are provided with the same demonstrations which are collected as described in Section 4.5. See Section 4.5

Behavior Cloning: Direct supervised learning on the offline suboptimal demonstrations.

Twin Delayed Deep Deterministic Policy Gradients (TD3) [11]: State of the art actor-critic algorithm which trains a deterministic policy to maximize a learned critic.

Soft Actor-Critic (SAC) [13]: State of the art actor-critic algorithm which trains a stochastic policy which maximizes a combination of the Q value of the policy and the expected entropy of the policy to encourage exploration.

Generalized Q Estimation (GQE): A complex return estimation method from [58] for actor-critic methods, which computes a weighted average over TD-i estimates for a range of i. GQE is implemented on top of SAC (see supplement for more details). We tune the range of horizons considered and the eligibility trace value (corresponding to λ in GAE).

Overcoming Exploration from Demonstrations (OEFD) [42]: OEFD builds on DDPG [36] by adding a loss which encourages imitating demonstrations and a learned filter which determines when to activate this loss. Our implementation does not include hindsight experience replay since it is not applicable to most of our environments.

Conservative Q Learning (CQL) [32]: A recent offline RL algorithm that addresses value overestimation with a conservative Q function. Here we also update CQL online after pre-training offline on demonstrations in order to provide a fair comparison with other algorithms.

Advantage Weighted Actor-Critic (AWAC) [41]: A recent offline reinforcement learning algorithm designed for fast online fine-tuning.

We also implement versions of each of the above RL algorithms with MCAC (TD3 + MCAC, SAC + MCAC, GQE + MCAC, OEFD + MCAC, CQL + MCAC, AWAC + MCAC).

The behavior cloning comparison serves to determine whether online learning is beneficial in general, while the other comparisons study whether MCAC can be used to accelerate reinforcement learning for commonly used actor-critic algorithms (TD3, SAC, and GQE, which is essentially SAC with complex returns) and for recent algorithms for RL from demonstrations (OEFD, CQL and AWAC).

Implementation Details

For all algorithms, all Q functions and policies are approximated using deep neural networks with 2 hidden layers of size 256. They are all updated using the Adam optimizer from [28].

Behavioral Cloning

We used a straightforward implementation of behavioral cloning, regressing with a mean square error loss. For all experiments learners were provided with the same number of demonstrators as the other algorithms and optimized for 10000 gradient steps using a learning rate of 1×10^{-4}.

Twin Delayed Deep Deterministic Policy Gradients

We use the author implementation of TD3 from [11], modifying it to implement MCAC. In order to maintain the assumption about complete trajectories described in the main text, we modify the algorithm to only add to the replay buffer at the end of sampled trajectories, but continue to update after each timestep. We found the default hyperparameters from the repository to be sufficient in all environments.

Soft Actor Critic

For all SAC experiments we used a modified version of the SAC implementation from [69] which implements SAC with a double-Q critic update function to combat Q overestimation. Additionally, we modify the algorithm to satisfy the trajectory assumption as in Section 4.5. We mostly use the default hyperparameters from [13], but tuned α and τ. Parameter choices are shown in Table 4.1.

Table 4.1: Hyperparameters for SAC

Parameter	Navigation	MuJoCo	Robosuite
Learning Rate	3×10^{-4}	3×10^{-4}	3×10^{-4}
Automatic Entropy Tuning	False	False	False
Batch Size	256	256	256
Hidden Layer Size	256	256	256
# Hidden Layers	2	2	2
# Updates Per Timestep	1	1	1
α	0.2	0.1	0.05
γ	0.99	0.99	0.99
τ	5×10^{-2}	5×10^{-3}	5×10^{-2}

Generalized Q Estimation

Similarly to the advantage estimation method in [58], we estimate Q values by computing a weighted average over a number of Q estimates estimated with k-step look-aheads. Concretely, if a $Q_t^{(k)}$ is a Q estimate with a k-step look-ahead given by

$$Q_t^{(k)} = \sum_{i=0}^{k-1} r_{t+i} + \gamma^k Q_\theta(s_{t+k}, a_{t+k}), \tag{4.5.1}$$

we compute the n-step GQE estimate Q_t^{GQE} as

$$Q_t^{\text{GQE}} = \frac{1-\lambda}{1-\lambda^n} \sum_{k=1}^{n} \lambda^{k-1} Q_t^{(k)}. \tag{4.5.2}$$

We built GQE on top of SAC, using the SAC Q estimates for the values of Q_θ. However, in principle this method can be applied to other actor-critic methods.

Where applicable we used the hyperparameters from SAC, and tuned the values of λ and n as hyperparameters, trying values in the sets $\lambda \in \{0.8, 0.9, 0.95, 0.99, 0.999\}$ and n values in the set $n \in \{8, 16, 32, 64\}$. The chosen parameters for each environment are given in Table 4.2.

Table 4.2: Hyperparameters for GQE

Parameter	Navigation	Extraction	Push	Door	Lift
λ	0.9	0.95	0.95	0.9	0.9
n	32	8	16	16	16

Overcoming Exploration with Demonstrations

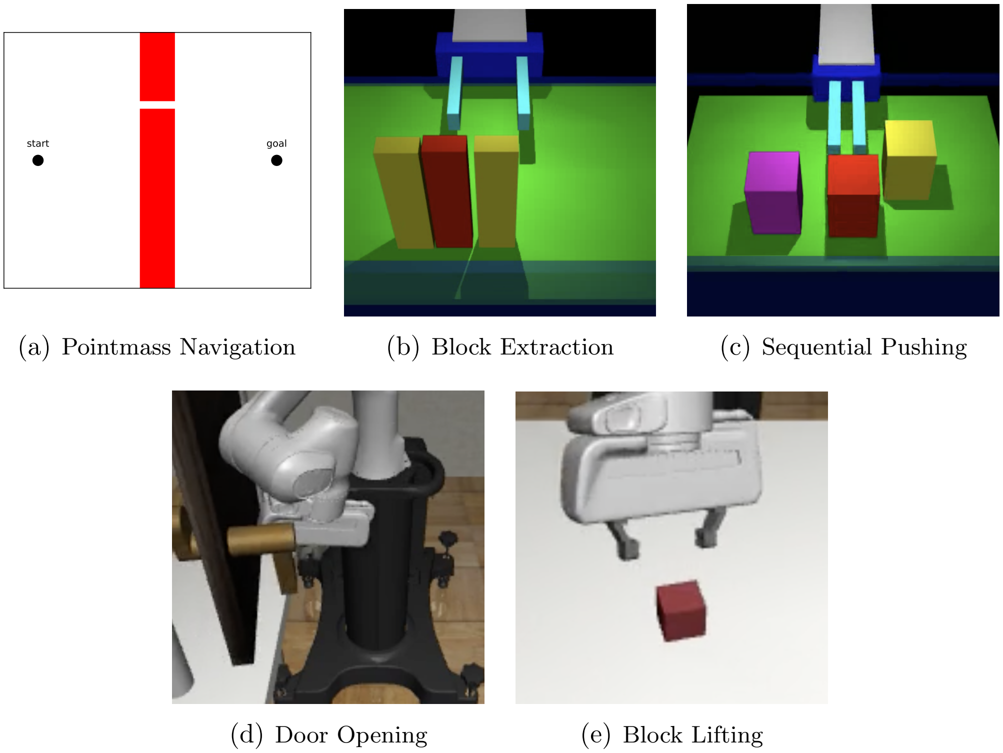
We implement the algorithm from [42] on top of the implementation of TD3 described in Section 4.5. Because it would provide an unfair advantage over comparisons, the agent is not given the ability to reset to arbitrary states in the replay buffer. Since our setting is not goal-conditioned, our implementation does not include hindsight experience replay. For the value λ balancing the actor critic policy update loss and behavioral cloning loss, we use $\lambda = 1.0$. In all experiments the agent is pretrained on offline data for 10000 gradient steps.

Conservative Q-learning

Offline reinforcement learning algorithm that produces a lower bound on the value of the current policy. We used the implementation from [**SAC˙CQL**], which implements CQL on top of SAC as is done in the original paper, modified for additional online-finetuning. We used the default hyperparameters from [32] in all environments and pretrained the agent on offline data for 10000 gradient steps.

Advantage Weighted Actor Critic

For AWAC experiments we use the implementation from [63], once again modifying it to maintain the assumption about complete trajectories and to implement MCAC. We found the default hyperparameter values to be sufficient in all settings. In all experiments the agent is pretrained on offline data for 10000 gradient steps.



(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing

(d) Door Opening (e) Block Lifting

Figure 4.1: MCAC Domains: We evaluate MCAC on five continuous control domains: a pointmass navigation environment, and four high-dimensional robotic control domains. All domains are associated with relatively unshaped reward functions, which only indicate constraint violation, task completion, or completion of a subtask.

Results

In Section 4.5, we study MCAC on a simple didactic environment to better understand how it affects the learned Q-values. We then study whether MCAC can be used to accelerate exploration on a number of continuous control domains. In Section 4.5, we study whether MCAC enables more efficient learning when built on top of widely used actor-critic RL algorithms (SAC, TD3, and GQE). Then in Section 4.5, we study whether MCAC can provide similar benefits when applied to recent RL from demonstration algorithms (OEFD, CQL, and AWAC). Additionally, in the supplement we provide experiments involving other baselines, investigating the sensitivity of MCAC to the quality and quantity of demonstration data, and investigating its sensitivity to pretraining.

MCAC Didactic Example

In order to better understand the way MCAC affects Q estimates, we visualize Q estimates when MCAC is applied to SAC after 50000 timesteps of training in the Pointmass Navigation environment, in Figure 4.2. Here we visualize Q-values for the entire replay buffer, including offline demonstrations,

When training without MCAC (top row), the agent is unable to learn a useful Q function and thus does not learn to complete the task (the only successful trajectories shown are offline data). However, even when this is the case, the MCAC estimate is able to effectively propagate reward signal backwards along the demonstrator trajectories, predicting higher rewards early on (top right). We see that the GQE estimates (top middle) are somewhat more effective than the Bellman ones at propagating reward, but not as effective as MCAC. When the agent is trained with MCAC (bottom row), the agent learns a useful Q function that it uses to reliably complete the task (bottom left). As expected with a high-performing policy, its Bellman estimates, GQE estimates and MCAC estimates are similar.

MCAC and Standard RL Algorithms

In Figure 4.3, we study the impact of augmenting SAC, TD3 and GQE with the MCAC target Q-function. Note that all methods, both with and without MCAC, we initialize their replay buffers with the same set of demonstrations. Results suggest that MCAC is able to accelerate learning for both TD3 and SAC across all environments, and is able to converge to performance either on-par with or better than the demonstrations. In the Pointmass Navigation and Block Lifting tasks, SAC and TD3 make no task progress without MCAC.

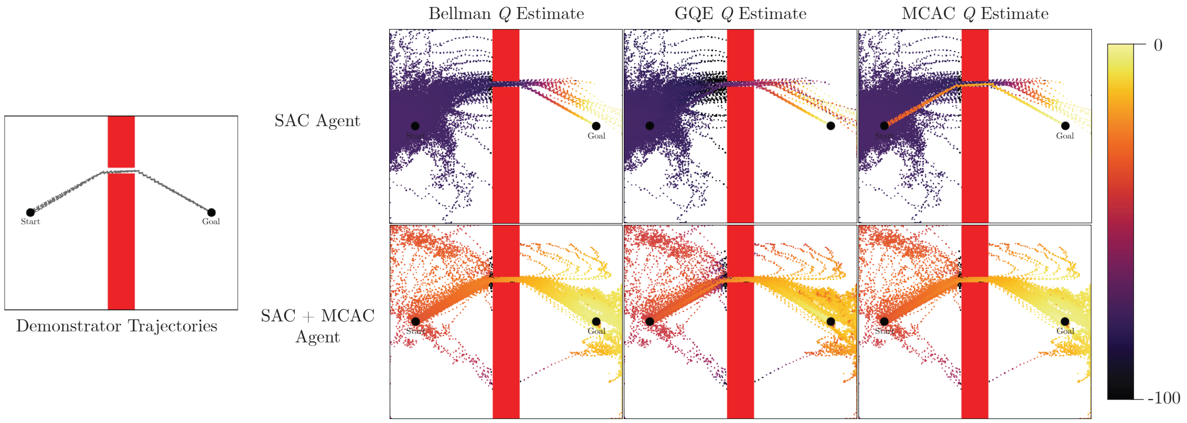
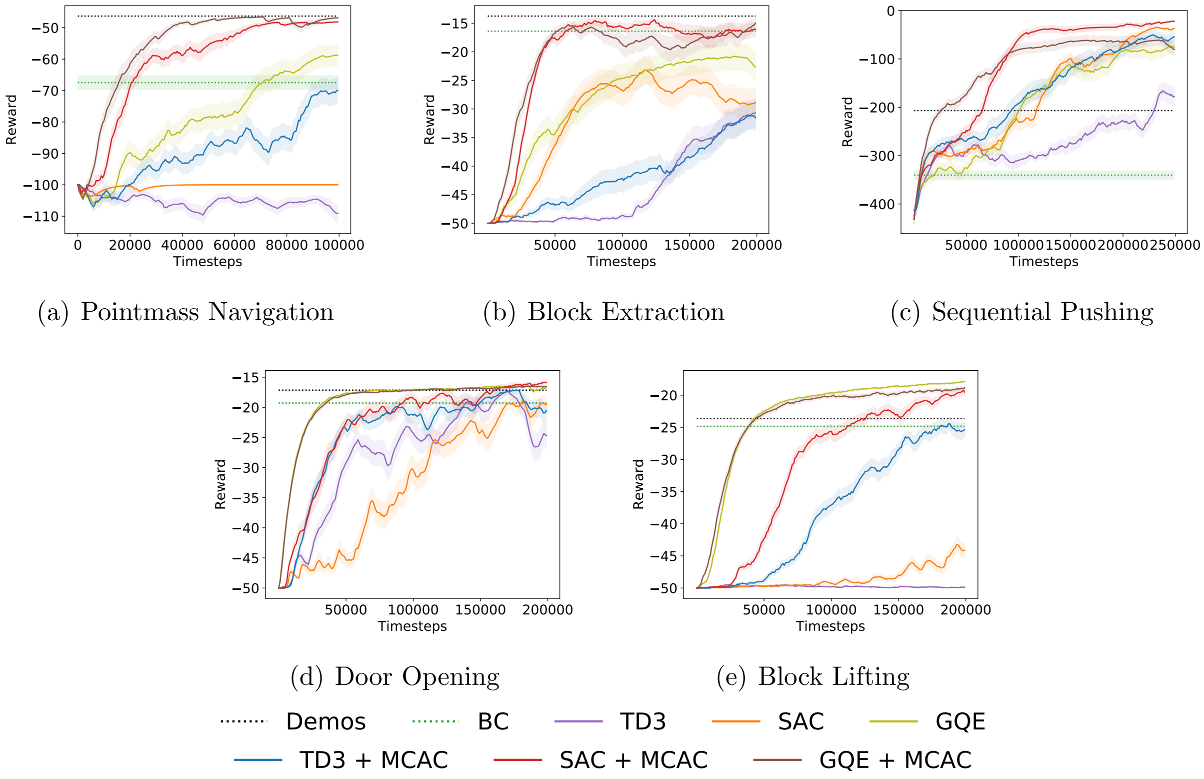


Figure 4.2: **MCAC Replay Buffer Visualization:** Scatter plots showing Bellman, GQE and MCAC Q estimates on the entire replay buffer, including offline demonstrations, for SAC learners with and without the MCAC modification after 50000 timesteps of training. The top row shows data and Q estimates obtained while training a baseline SAC agent without MCAC, while the bottom row shows the same when SAC is trained with MCAC. The left column shows Bellman Q estimates on each replay buffer sample while the middle column shows GQE estimates and the right column shows MCAC estimates. Results suggest that MCAC is helpful for propagating rewards along demonstrator trajectories.



(a) Pointmass Navigation

(b) Block Extraction

(c) Sequential Pushing

(d) Door Opening

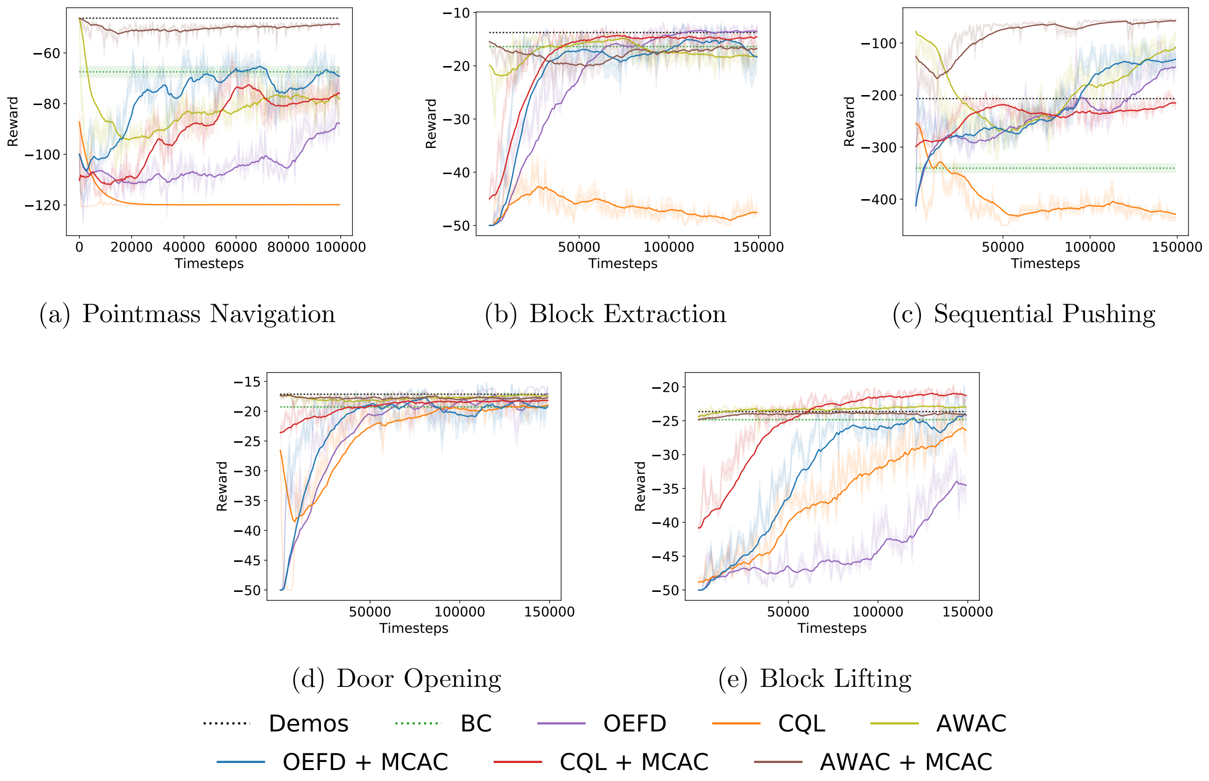
(e) Block Lifting

Figure 4.3: **MCAC and Standard RL Algorithms Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. We find that MCAC improves the learning efficiency of TD3, SAC, and GQE across all 5 environments.

MCAC also accelerates learning for GQE for the Pointmass Navigation, Block Extraction, and Sequential Pushing environments. In the Door Opening and Block Lifting environments, MCAC leaves performance largely unchanged since GQE already achieves performance on par with the next best algorithm without MCAC.

MCAC and RL From Demonstrations Algorithms

In Figure 4.4, we study the impact of augmenting OEFD [42], CQL [32] and AWAC [41] with the MCAC target Q-function. Results suggest that MCAC improves the learning efficiency of OEFD on the Pointmass Navigation, Sequential Pushing, and Block Lifting tasks, but does not have a significant positive or negative affect on performance for the Block Extraction and Door Opening tasks. MCAC improves the performance of AWAC on the Pointmass Navigation and Sequential Pushing environments, stabilizing learning while the versions without MCAC see performance fall off during online fine tuning. On the other 3 environments where AWAC is able to immediately converge to a stable policy after offline pre-training,



(a) Pointmass Navigation (b) Block Extraction (c) Sequential Pushing

(d) Door Opening (e) Block Lifting

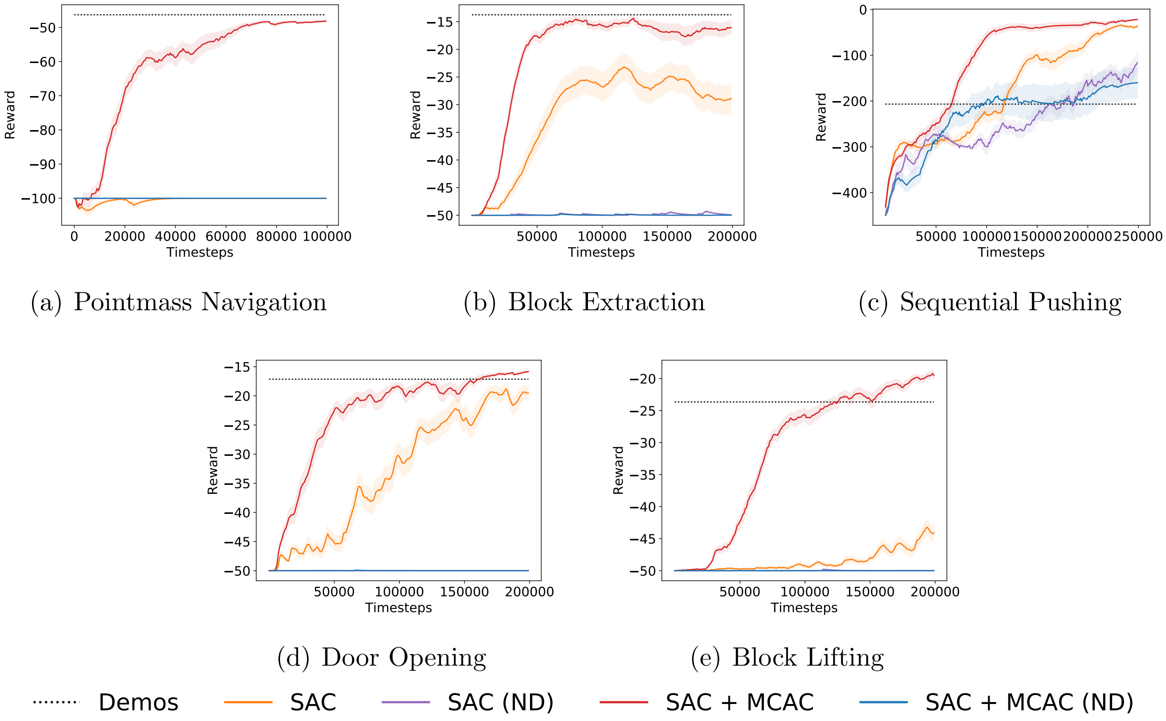
Figure 4.4: **MCAC and RL from Demonstrations Algorithm Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. When OEFD or AWAC achieve high performance almost immediately, MCAC has little impact on performance. However, when OEFD and AWAC are unable to learn efficiently, MCAC accelerates and stabilizes policy learning.

MCAC has no significant negative effect on its performance. In all tasks, MCAC improves the performance of CQL. In particular, for the Pointmass Navigation, Block Extraction and Sequential Pushing tasks, CQL makes almost no progress while the version with MCAC learns to complete the task reliably.

4.6 Additional Experiments

No Demonstrations

To study the effects that MCAC has without demonstration data in the replay buffer we compare performance with and without MCAC and demonstrations in all environments with the SAC learner, shown in Figure 4.5. Overall we see that, as desired, the agent is unable to make progress in most environments. The only exception is the sequential pushing environment results (Figure 4.5(c)), where the intermediate reward for pushing each block helps the agent learn to make some progress. Overall, this experiment does not conclusively answer whether MCAC is helpful without demonstrations, but this is an exciting direction for future work.



(a) Pointmass Navigation

(b) Block Extraction

(c) Sequential Pushing

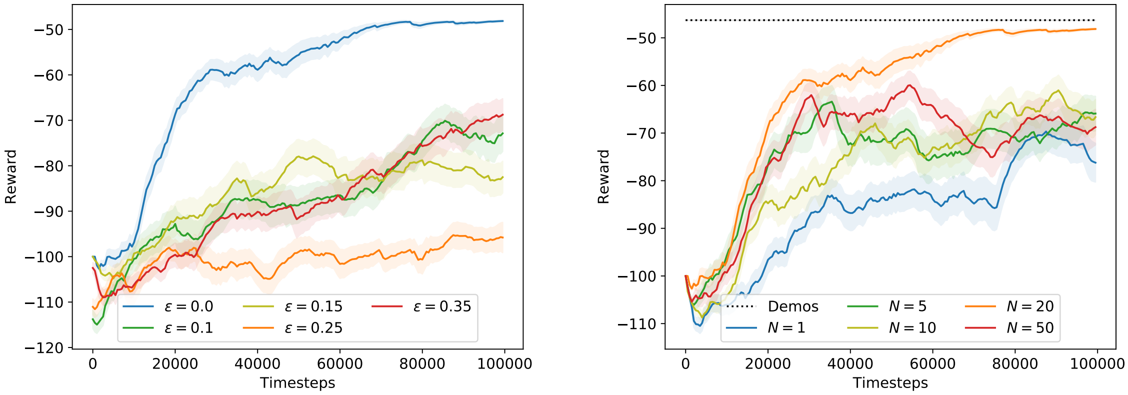
(d) Door Opening

(e) Block Lifting

Figure 4.5: **MCAC without Demonstrations:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds for experiments with demonstrations and 3 seeds for experiments without them. We see that in all environments the demonstrations are critical for learning an optimal policy. The only place the variants without demonstrations make progress is in the push environment, because of the intermediate reward for pushing each block.

MCAC Sensitivity Experiments

In Figure 4.6, we first study the impact of demonstration quality (Figure 4.6(a)) and quantity (Figure 4.6(b)) on MCAC when applied to SAC (SAC + MCAC) on the Pointmass Navigation domain. We evaluate sensitivity to demonstration quality by injecting ϵ-greedy noise into the demonstrator for the Pointmass Navigation domain. Results suggest that MCAC is somewhat sensitive to demonstration quality, since MCAC's performance does drop significantly for most values of ϵ, although it still typically makes some task progress. In Figure 4.6(b), results suggest that MCAC is relatively robust to the number of demonstration.



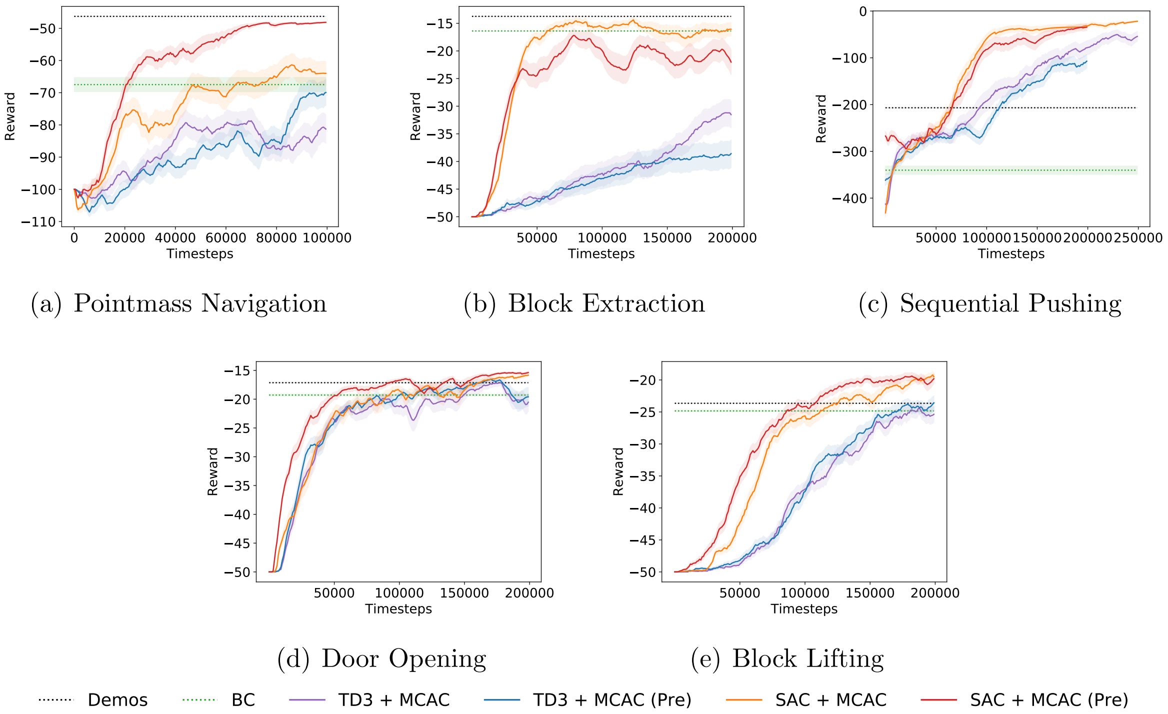
(a) Demonstration Quality

(b) Demonstration Quantity

Figure 4.6: **MCAC Sensitivity Experiments:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds for varying demonstration qualities (a) and quantities (b) for SAC + MCAC. (a): Results suggest that MCAC is somewhat sensitive to demonstration quality, as when ϵ-greedy noise is injected into the demonstrator, MCAC's performance does drop significantly, although it eventually make some task progress for most values of ϵ. (b): MCAC appears to be much less sensitive to demonstration quantity, and is able to achieve relatively high performance even with a single task demonstration.

Pretraining

In Figure 4.7 we ablate on the pretraining step of the algorithm to determine whether it is useful to include. We find that it was helpful for the Navigation environment but unhelpful and sometimes limiting in other settings. Thus, we leave pretraining as a hyperparameter to be tuned.



(a) Pointmass Navigation

(b) Block Extraction

(c) Sequential Pushing

(d) Door Opening

(e) Block Lifting

Figure 4.7: **MCAC with and without Pretraining Results:** Learning curves showing the exponentially smoothed (smoothing factor $\gamma = 0.9$) mean and standard error across 10 random seeds. We find that other than in the navigation environment pretraining does not provide a significant benefit.

Chapter 5

Conclusions and Limitations

In this dissertation we propose the use of offline demonstrations as a means to guide online RL exploration, and present two novel algorithms using this to improve efficiency and safety during learning.

In Chapter 3, we present LS^3, a scalable algorithm for safe and efficient policy learning for visuomotor tasks. LS^3 structures exploration by learning a safe set in a learned latent space, which captures the set of states from which the agent is confident in task completion. LS^3 then ensures that the agent can plan back to states in the safe set, encouraging consistent task completion during learning. Experiments suggest that LS^3 can safely and efficiently learn 4 visuomotor control tasks, including a challenging sequential pushing task in simulation and a cable routing task on a physical robot.

While LS^3 did show encouraging results in the domains we experimented in, it also had numerous limitations. For one, we found the algorithm to require more offline dataset engineering than would be desirable for widespread deployment. We also found the dynamics to fail in more complicated environments than those we present. Both of these issues need to be addressed before it is ready for widespread use.

In Chapter 4, we present Monte Carlo augmented Actor-Critic (MCAC), a simple, yet highly effective, change that can be applied to any actor-critic algorithm in order to accelerate reinforcement learning from demonstrations for sparse reward tasks. We present empirical results suggesting that MCAC often significantly improves performance when applied to three state-of-the-art actor-critic RL algorithms and three RL from demonstrations algorithms on five different continuous control domains.

Despite strong empirical performance, MCAC also has some limitations. We found that while encouraging higher Q-value estimates is beneficial for sparse reward tasks, when rewards are dense and richly informative, MCAC is not helpful and can even hinder learning by overestimating Q-values. From an ethical standpoint, reinforcement learning algorithms such as MCAC can be used to automate aspects of a variety of interactive systems, such as robots, online retail systems, or social media platforms. However, poor performance when policies haven't yet converged, or when hand-designed reward functions do not align with true human intentions, can cause significant harm to human users. Lastly, while MCAC shows convincing empirical performance, its theoretical backbone is questionable, since it relies on using n-step rollouts, on-policy data, from past policies in the replay buffer.

Chapter 6

Future Work

There are a wide variety of directions we hope to extend the work in this dissertation in the future.

For Latent Space Safe Sets (LS^3), we list some future directions below:

- For computational reasons, we chose to pretrain the latent embeddings on the offline dataset and then freeze the encoders for use online. While this was beneficial for the first iteration of the project where fast computation was critical, it was limiting because it left the agent unable to learn to deal with observations it doesn't observe in the offline dataset. In future iterations of the algorithm it would be beneficial to improve the pipeline to update the encoder online.
- LS^3 is built on a variety of older pieces of technology, and would likely benefit by replacing these with more advanced algorithms. For example, while β-VAE learned a sufficient representation for the domains we considered, a version of LS^3 learned on top of masked autoencoders [17] would likely be applicable to a wider variety of situations. It would also be interesting to explore how LS^3's ideas combine with more advanced world models, such as that from DreamerV3 [16].
- Assuming the changes above lead to concrete improvements, it would be interesting to experiment with LS^3 in more complex and long horizon simulated and physical environments.

There are also a variety of future directions we hope to investigate while extending MCAC.

- As mentioned before, MCAC achieves impressive empirical results with the downside that it loses its theoretical convergence guarantees. A key issue making this difficult is that the use of n-step returns implicitly assumes that the data is on-policy, but because that data is coming from a replay buffer, this is not the case. It would be interesting to investigate whether there is a way to reason about this mixture, or whether it is possible to change the algorithm to get the same or better empirical performance while providing better theoretical guarantees.

- While MCAC performed well in the domains we considered, it would be interesting to explore its applicability to a wider variety of longer-horizon tasks. Down the road, it would also be useful to run experiments deploying it on physical robots to see whether its benefits are present in that setting as well.
- While policy learning, which we considered in this project, is the most obvious application of the improved MCAC Q target estimation step, there are a variety of other applications we hope to investigate MCAC in. For example, it would be interesting to see whether it improves safety when applied to the safety critics learned in [67, 73].

Bibliography

[1] Brenna D Argall et al. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[2] Somil Bansal et al. "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances". In: *Conference on Decision and Control (CDC)*. 2017.

[3] Marc Bellemare et al. "Unifying count-based exploration and intrinsic motivation". In: *Advances in Neural Information Processing Systems*. 2016, pp. 1471–1479.

[4] Douglas A Bristow, Marina Tharayil, and Andrew G Alleyne. "A survey of iterative learning control". In: *IEEE control systems magazine* (2006).

[5] K. Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Proc. Advances in Neural Information Processing Systems*. 2018.

[6] Kurtland Chua. *Experiment code for "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models"*. `https://github.com/kchua/handful-of-trials`. 2018.

[7] Kurtland Chua et al. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models". In: *Proc. Advances in Neural Information Processing Systems*. 2018.

[8] MP. Deisenroth and CE. Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proc. Int. Conf. on Machine Learning*. 2011.

[9] Frederik Ebert et al. "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control". In: *arXiv preprint arXiv:1812.00568* (2018).

[10] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*. 2016.

[11] Scott Fujimoto, Herke van Hoof, and Dave Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *Proc. Int. Conf. on Machine Learning*. 2018.

[12] Yang Gao et al. "Reinforcement Learning from Imperfect Demonstrations". In: *CoRR* abs/1802.05313 (2018). arXiv: `1802.05313`. URL: `http://arxiv.org/abs/1802.05313`.

[13] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proc. Int. Conf. on Machine Learning* (2018).

[14] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proc. Int. Conf. on Machine Learning.*

[15] Danijar Hafner et al. “Learning Latent Dynamics for Planning from Pixels”. In: *Proc. Int. Conf. on Machine Learning* (2019).

[16] Danijar Hafner et al. “Mastering Diverse Domains through World Models”. In: *arXiv preprint arXiv:2301.04104* (2023).

[17] Kaiming He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* June 2022, pp. 16000–16009.

[18] Todd Hester et al. “Deep q-learning from demonstrations”. In: *Thirty-Second AAAI Conference on Artificial Intelligence.* 2018.

[19] Irina Higgins et al. “beta-vae: Learning basic visual concepts with a constrained variational framework”. In: *Proc. Int. Conf. on Learning Representations* (2017).

[20] Jeffrey Ichnowski et al. “Deep learning can accelerate grasp-optimized motion planning”. In: *Science Robotics* 5.48 (2020).

[21] B. Ichter and M. Pavone. “Robot Motion Planning in Learned Latent Spaces”. In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2407–2414. DOI: `10.1109/LRA.2019.2901898`.

[22] Greg Kahn et al. “Uncertainty-Aware Reinforcement Learning for Collision Avoidance”. In: *CoRR* (2017).

[23] Dmitry Kalashnikov et al. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *Conf. on Robot Learning (CoRL)* (2018).

[24] Bingyi Kang, Zequn Jie, and Jiashi Feng. “Policy optimization with demonstrations”. In: *International Conference on Machine Learning.* PMLR. 2018, pp. 2469–2478.

[25] Peter Kazanzides et al. “An Open-Source Research Kit for the da Vinci Surgical System”. In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA).* 2014.

[26] Peter Kazanzides et al. “An open-source research kit for the da Vinci® Surgical System”. In: *2014 IEEE international conference on robotics and automation (ICRA).* IEEE. 2014, pp. 6434–6439.

[27] Beomjoon Kim et al. “Learning from Limited Demonstrations.” In: *NIPS.* Citeseer. 2013, pp. 2859–2867.

[28] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2015).

[29] Jens Kober and Jan Peters. "Policy search for motor primitives in robotics". In: *Learning Motor Skills*. Springer, 2014, pp. 83–117.

[30] George Konidaris, Scott Niekum, and Philip S Thomas. "Td_gamma: Re-evaluating complex backups in temporal difference learning". In: *Advances in Neural Information Processing Systems* 24 (2011), pp. 2402–2410.

[31] Victoria Krakovna et al. "Specification gaming: the flip side of AI ingenuity". In: *DeepMind Blog* (2020).

[32] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 1179–1191. URL: `https://proceedings.neurips.cc/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf`.

[33] Arsenii Kuznetsov et al. "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5556–5566.

[34] Michael Laskin et al. "Reinforcement Learning with Augmented Data". In: (2020). arXiv:2004.14990.

[35] Ian Lenz, Ross A. Knepper, and Ashutosh Saxena. "DeepMPC: Learning Deep Latent Features for Model Predictive Control". In: *Robotics: Science and Systems*. 2015.

[36] Timothy P. Lillicrap et al. "Continuous control with deep reinforcement learning". In: *CoRR* abs/1509.02971 (2015). arXiv: `1509.02971`. URL: `http://arxiv.org/abs/1509.02971`.

[37] Martina Lippi et al. "Latent Space Roadmap for Visual Action Planning of Deformable and Rigid Object Manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2020.

[38] Kendall Lowrey et al. "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control". In: *Proc. Int. Conf. on Machine Learning*. 2019.

[39] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[40] Anusha Nagabandi et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2018.

[41] Ashvin Nair et al. *AWAC: Accelerating Online Reinforcement Learning with Offline Datasets*. 2021. arXiv: `2006.09359 [cs.LG]`.

[42] Ashvin Nair et al. "Overcoming Exploration in Reinforcement Learning with Demonstrations". In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)* (2018).

[43] Ashvin Nair et al. "Visual reinforcement learning with imagined goals". In: *Proc. Advances in Neural Information Processing Systems* (2018).

[44] Suraj Nair, Silvio Savarese, and Chelsea Finn. "Goal-Aware Prediction: Learning to Model What Matters". In: *Proceedings of the 37th International Conference on Machine Learning*. 2020, pp. 7207–7219.

[45] Georg Ostrovski et al. "Count-based exploration with neural density models". In: *International conference on machine learning*. PMLR. 2017, pp. 2721–2730.

[46] Xue Bin Peng et al. *Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning*. 2019. arXiv: `1910.00177 [cs.LG]`.

[47] Vitchyr H Pong et al. "Skew-fit: State-covering self-supervised reinforcement learning". In: *Proc. Int. Conf. on Machine Learning* (2020).

[48] S. Quinlan and O. Khatib. "Elastic bands: connecting path planning and control". In: *International Conference on Robotics and Automation*. 1993, 802–807 vol.2.

[49] Aravind Rajeswaran et al. *Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations*. 2018. arXiv: `1709.10087 [cs.LG]`.

[50] Spencer M Richards, Felix Berkenkamp, and Andreas Krause. "The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems". In: *Conference on Robot Learning*. PMLR. 2018, pp. 466–476.

[51] Ugo Rosolia and Francesco Borrelli. "Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework". In: *IEEE Transactions on Automatic Control* (2018).

[52] Ugo Rosolia and Francesco Borrelli. "Sample-Based Learning Model Predictive Control for Linear Uncertain Systems". In: *CoRR* abs/1904.06432 (2019). URL: `http://arxiv.org/abs/1904.06432`.

[53] Ugo Rosolia, Xiaojing Zhang, and Francesco Borrelli. "A Stochastic MPC Approach with Application to Iterative Learning". In: *2018 IEEE Conference on Decision and Control (CDC)* (2018).

[54] Reuven Rubinstein. "The cross-entropy method for combinatorial and continuous optimization". In: *Methodology and computing in applied probability* 1.2 (1999), pp. 127–190.

[55] Andrei A Rusu et al. "Sim-to-real robot learning from pixels with progressive nets". In: *Conference on Robot Learning*. PMLR. 2017, pp. 262–270.

[56] Stefan Schaal et al. "Learning from demonstration". In: *Advances in neural information processing systems* (1997), pp. 1040–1046.

[57] Gerrit Schoettler et al. "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards". In: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (2020).

[58] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *International Conference on Learning Representations (ICLR)* (June 2016).

[59] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *CoRR* abs/1707.06347 (2017). arXiv: `1707.06347`. URL: `http://arxiv.org/abs/1707.06347`.

[60] John Schulman et al. “Trust Region Policy Optimization”. In: Proceedings of Machine Learning Research 37 (July 2015). Ed. by Francis Bach and David Blei, pp. 1889–1897. URL: `http://proceedings.mlr.press/v37/schulman15.html`.

[61] D. Seita et al. “Fast and Reliable Autonomous Surgical Debridement with Cable-Driven Robots Using a Two-Phase Calibration Procedure”. In: *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*. 2018.

[62] Sahil Sharma et al. “Learning to Mix n-Step Returns: Generalizing lambda-Returns for Deep Reinforcement Learning”. In: 2018. arXiv: `1705.07445 [cs.AI]`.

[63] Harshit Sikchi and Albert Wilcox. *pytorch-AWAC*. URL: `https://github.com/hari-sikchi/AWAC`.

[64] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.

[65] Avi Singh et al. “End-to-end robotic reinforcement learning without reward engineering”. In: *Proc. Robotics: Science and Systems (RSS)* (2019).

[66] Aravind Srinivas et al. “Universal Planning Networks”. In: *Proc. Int. Conf. on Machine Learning* (Apr. 2018).

[67] Krishnan Srinivasan et al. “Learning to be Safe: Deep RL with a Safety Critic”. In: *arXiv preprint arXiv:2010.14603* (2020).

[68] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 1st. Cambridge, MA, USA: MIT Press, 1998. ISBN: 0262193981.

[69] Pranjal Tandon. *pytorch-soft-actor-critic*. URL: `https://github.com/pranz24/pytorch-soft-actor-critic`.

[70] Yuval Tassa et al. *dm-control: Software and Tasks for Continuous Control*. 2020. arXiv: `2006.12983 [cs.RO]`.

[71] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. “Reward Constrained Policy Optimization”. In: *Proc. Int. Conf. on Learning Representations*. 2019.

[72] Brijen Thananjeyan et al. *ABC-LMPC: Safe Sample-Based Learning MPC for Stochastic Nonlinear Dynamical Systems with Adjustable Boundary Conditions*. 2020.

[73] Brijen Thananjeyan et al. “Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones”. In: *NeurIPS Deep Reinforcement Learning Workshop* (2020).

[74] Brijen Thananjeyan et al. “Safety Augmented Value Estimation From Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3612–3619.

[75] Stephen Tian et al. “Model-Based Visual Planning with Self-Supervised Functional Distances”. In: *Proc. Int. Conf. on Learning Representations* (2021).

[76] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on.* IEEE. 2012, pp. 5026–5033. URL: `https://ieeexplore.ieee.org/abstract/document/6386109/`.

[77] Hado Van Hasselt, Arthur Guez, and David Silver. "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI conference on artificial intelligence.* Vol. 30. 1. 2016.

[78] Hado Van Hasselt et al. *Deep Reinforcement Learning and the Deadly Triad.* 2018. arXiv: `1812.02648 [cs.LG]`.

[79] Matej Vecerik et al. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *CoRR* abs/1707.08817 (2017).

[80] Albert Wilcox et al. "Monte Carlo Augmented Actor-Critic for Sparse Reward Deep Reinforcement Learning from Suboptimal Demonstrations". In: *Advances in Neural Information Processing Systems.* Ed. by Alice H. Oh et al. 2022. URL: `https://openreview.net/forum?id=FLzTj4ia8BN`.

[81] Albert Wilcox* et al. "LS3: Latent Space Safe Sets for Long-Horizon Visuomotor Control of Sparse Reward Iterative Tasks". In: *Conference on Robot Learning (CoRL).* PMLR. 2021.

[82] Robert Wright et al. "Exploiting Multi-step Sample Trajectories for Approximate Value Iteration". In: *Machine Learning and Knowledge Discovery in Databases.* Ed. by Hendrik Blockeel et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 113–128. ISBN: 978-3-642-40988-2.

[83] Robert William Wright et al. "CFQI: Fitted Q-Iteration with Complex Returns." In: *AAMAS.* Citeseer. 2015, pp. 163–170.

[84] Zheng Wu et al. "Learning Dense Rewards for Contact-Rich Manipulation Tasks". In: *arXiv preprint arXiv:2011.08458* (2020).

[85] Jiexin Xie et al. "Deep reinforcement learning with optimized reward functions for robotic trajectory planning". In: *IEEE Access* 7 (2019), pp. 105669–105679.

[86] Jesse Zhang et al. "Cautious adaptation for reinforcement learning in safety-critical settings". In: *International Conference on Machine Learning.* PMLR. 2020, pp. 11055–11065.

[87] Marvin Zhang et al. "Solar: Deep structured representations for model-based reinforcement learning". In: *International Conference on Machine Learning.* PMLR. 2019, pp. 7444–7453.

[88] Yuke Zhu et al. "robosuite: A Modular Simulation Framework and Benchmark for Robot Learning". In: *arXiv preprint arXiv:2009.12293.* 2020.

[89] Zhaoxuan Zhu et al. In: (2021). arXiv: `2105.11640 [cs.LG]`.