LS^3: Latent Space Safe Sets for Long-Horizon Visuomotor Control of Sparse Reward Iterative Tasks

Anonymous Author(s)
Affiliation
Address
email

Abstract: Reinforcement learning (RL) has shown impressive success in exploring high-dimensional environments to learn complex tasks, but can often exhibit unsafe behaviors and require extensive environment interaction when exploration is unconstrained. 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 observation spaces is exceedingly challenging. We present a novel continuous representation for safe sets framed as *a binary classification problem* in a learned latent space, which flexibly scales to high-dimensional image observations. We then present a new algorithm, Latent Space Safe Sets (LS^3), which uses this representation for long-horizon control. We evaluate LS^3 on 4 domains, including a challenging sequential pushing task in simulation and a physical cable routing task. We find that LS^3 can use prior task successes to restrict exploration and learn more efficiently than prior algorithms while satisfying constraints. See `https://tinyurl.com/latent-safe-sets` for supplementary material.

Keywords: Reinforcement Learning, Imitation Learning, Safety

1 Introduction

Visual planning over learned forward dynamics models is a popular area of research in robotic control from images [1, 2, 3, 4, 5, 6, 7], as it enables closed-loop, model-based control for tasks where the state of the system is not directly observable or difficult to analytically model, such as the configuration of a sheet of fabric or segment of cable. These methods learn predictive models over either images or a learned latent space, which can then be used to plan actions which maximize some task reward. While these approaches have significant promise, there are several open challenges in learning policies from visual observations. First, reward specification is particularly challenging for visuomotor control tasks, because high-dimensional observations often do not expose the necessary features required to design dense, informative reward functions [8], especially for long-horizon tasks. Second, while many prior reinforcement learning methods have been successfully applied to image-based control tasks [9, 10, 11, 12, 13], learning policies from image observations often requires extensive exploration due to the high dimensionality of the observation space and the difficulties in reward specification, making safe and efficient learning exceedingly challenging.

One promising strategy for efficiently learning safe control policies is to learn a safe set [14, 15], 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 [16, 17, 14], 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 [14, 18, 16], using them for controller design for practical problems requires developing continuous approximations at the expense of maintaining theoretical guarantees [17]. This choice of continuous approximation is a key element in determining the applications to which these safe sets can be used for control.

Submitted to the 5th Conference on Robot Learning (CoRL 2021). Do not distribute.

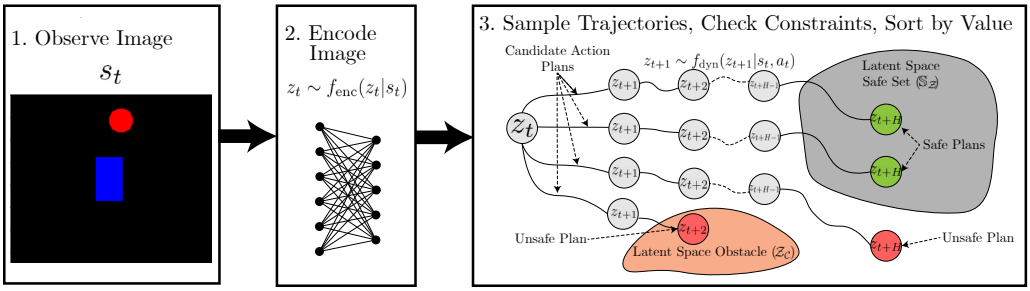

Figure 1: **Latent Space Safe Sets (LS^3):** At time t, LS^3 observes an image s_t of the environment. The image is first encoded to a latent vector $z_t \sim f_{\text{enc}}(z_t|s_t)$. Then, LS^3 uses a sampling-based optimization procedure to optimize H-length action sequences by sampling H-length latent 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.

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 [14] or via kernel density estimation with a tophat kernel function [17]. 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 [16, 14, 19].

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 Latent Space Safe Sets (LS^3), 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 LS^3 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 LS^3 can learn more efficiently than prior algorithms while consistently completing the task and satisfying constraints during learning.

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 [20]. Rosolia et al. [21], Rosolia and Borrelli [18, 14] 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, Rosolia et al. [21], Rosolia and Borrelli [18, 14] 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. Thananjeyan et al. [17] 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. Richards et al. [15] 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 [22, 23, 24, 25, 26, 27]. 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 Quinlan and Khatib [28] and Ichnowski et al. [29], which use model-based control with known dynamics to refine initially suboptimal motion plans, and Fu et al. [24], which uses demonstrations to seed a learned dynamics model for fast online adaptation using iLQR [24]. Thananjeyan et al. [17], Zhu et al. [30] present ILC algorithms which rapidly improve upon suboptimal demonstrations when system dynamics are unknown. However, these algorithms either require knowledge of system dynamics [28, 29] or are limited to low-dimensional state spaces [24, 17, 30] 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 [1, 31]. 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 [32, 33, 10, 34, 12, 13, 1, 35, 31]. 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 [2, 31, 35] and goal-conditioned reinforcement learning [13, 10]. However, safe reinforcement learning for visuomotor tasks has received substantially less attention. Thananjeyan et al. [36] and Kahn et al. [37] present reinforcement learning algorithms which estimate the likelihood of constraint violations to avoid them [36] or reduce the robot's velocity [37]. 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.

3 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, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ maps a state and action to a probability distribution over subsequent states, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ 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 Thananjeyan et al. [17], 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 [36]. 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

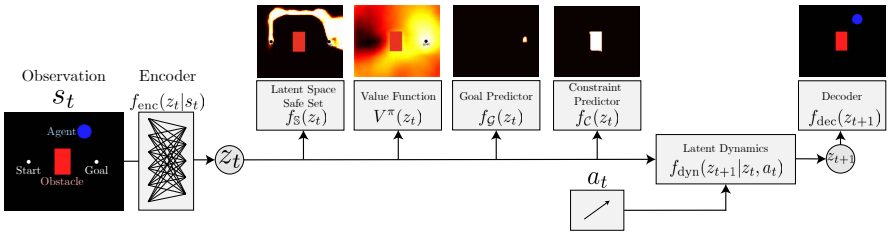

Figure 2: **LS^3 Learned Models**: LS^3 learns a low-dimensional latent representation of image-observations (Section 4.1) and learns a dynamics model, value function, reward function, constraint classifier, and safe set for constrained planning and task-completion driven exploration in this learned latent space. These models are then used for model-based planning to maximize the total value of predicted latent states (Section 4.3) while enforcing the safe set (Section 4.2) and user-specified constraints (Section 4.3).

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 $\pi : \mathcal{S} \to \mathcal{A}$, 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 $P_C^{\pi}(s)$ 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:

$$\pi^* = \arg\max_{\pi \in \Pi} \{R^{\pi} : \mathbb{E}_{s_0 \sim \mu}\left[P_C^{\pi}(s_0)\right] \leq \delta_{\mathcal{C}}\} \tag{1}$$

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 [36], $\mathcal{D}_{\text{constraint}}$ 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.

4 Latent Space Safe Sets (LS^3)

We describe how LS^3 uses demonstrations and online interaction to safely learn iteratively improving policies. Section 4.1 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 [26] in the learned latent space and models to estimate whether plans will likely complete the task (Section 4.2) and to estimate future rewards and constraint violations (Section 4.3) from predicted trajectories. In Section 4.4, 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 4.2 and 4.3) after every K rollouts. See Algorithm 1 and the supplement for further details on training procedures and data collection.

Algorithm 1 Latent Space Safe Sets (LS^3)

Require: offline dataset $\mathcal{D}$, number of updates U
Train VAE encoder f_{enc} and decoder f_{dec} (Section 4.1) using data from $\mathcal{D}$
Train dynamics f_{dyn}, safe set classifier $f_{\mathbb{S}}$(Section 4.2), and the value function V goal indicator $f_{\mathcal{G}}$, and constraint estimator $f_{\mathcal{C}}$ (Section 4.3) 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 μ.
    for $t \in \{1, \ldots, T\}$ **do**
      Choose and execute a_t (Section 4.4)
      Observe s_{t+1}, reward r_t, constraint c_t.
      $\mathcal{D} := \mathcal{D} \cup \{(s_t, a_t, s_{t+1}, r_t, c_t)\}$
  Update f_{dyn}, V, $f_{\mathcal{G}}$, $f_{\mathcal{C}}$, and $f_{\mathbb{S}}$ with data from $\mathcal{D}$.

4.1 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 [2, 35, 6, 38, 39, 31]. To learn such a representation, we train a β-variational autoencoder [40] 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 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 Laskin et al. [41], 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 [2] and found that varying d did not significantly affect performance.

4.2 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 4.1 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: $\mathbb{S}^j_{\mathcal{Z}} = \{z_t | f_{\mathbb{S}}(\cdot)(z_t) \geq \delta_{\mathbb{S}}\}$.

4.3 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 [17, 16, 14, 8], we train a recursively-defined value function (details in the supplement). Similar to the reward function, we use the encoder (Section 4.1) to train a classifier $f_{\mathcal{C}} : \mathcal{Z} \to [0, 1]$ 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.

4.4 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 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:

$$\underset{a_{t:t+H-1} \in \mathcal{A}^H}{\arg\max} \quad \mathbb{E}_{z_{t:t+H}} \left[\sum_{i=1}^{H-1} f_{\mathcal{G}}(z_{t+i}) + V^{\pi}(z_{t+H}) \right] \tag{2}$$

$$\text{s.t.} \quad z_t \sim f_{\text{enc}}(z_t|s_t) \tag{3}$$

$$z_{k+1} \sim f_{\text{dyn}}(z_{k+1}|z_k, a_k) \; \forall k \in \{t, \ldots, t+H-1\} \tag{4}$$

$$\hat{\mathbb{P}}\left(z_{t+H} \in \mathbb{S}^{j-1}_{\mathcal{Z}}\right) \geq 1 - \delta_{\mathbb{S}} \tag{5}$$

$$\hat{\mathbb{P}}(z_{t+i} \in \mathcal{Z}_{\mathcal{C}}) \leq \delta_{\mathcal{C}} \; \forall i \in \{0, \ldots, H-1\} \tag{6}$$

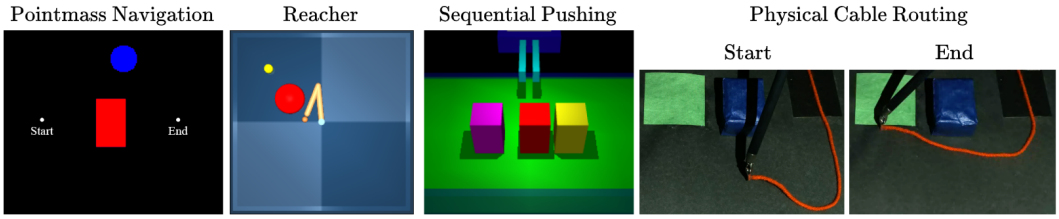

Figure 3: **Experimental Domains:** LS^3 is evaluated on 3 long-horizon, image-based, simulation environments: a visual navigation domain where the goal is to navigate the blue point mass to the right goal set while avoiding the red obstacle, a 2 degree of freedom reacher arm where the task is to navigate 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.

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) [42] which is a popular optimizer in model-based RL [43, 17, 16, 44, 36].

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 2). First, the current state s_t is encoded to z_t (equation 3). Then, for a candidate sequence of actions $a_{t:t+H-1}$, an H-step latent trajectory $\{z_{t+1}, \ldots, z_{t+H}\}$ is sampled from the learned dynamics f_{dyn} (equation 4). LS^3 constrains exploration using two chance constraints: (1) the terminal latent state in the plan must fall in the safe set (equation 5) and (2) all latent states in the trajectory must satisfy user-specified state-space constraints (equation 6). $\mathcal{Z}_{\mathcal{C}}$ 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.

5 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) [45]. Safe RL is of particular interest for surgical robots such as the dVRK due to its delicate structure, motivating safety, and relatively imprecise controls [17, 46], 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.

5.1 Comparisons

We evaluate LS^3 in comparison to prior algorithms that behavior clone suboptimal demonstrations before exploring online (**SACfD**) [47] or leverage offline reinforcement learning to learn a policy using all offline data before updating the policy online (**AWAC**) [48]. For both of these comparisons we enforce constraints via a tuned reward penalty of λ for constraint violations as in [49]. We also implement a version of SACfD with a learned recovery policy (**SACfD+RRL**) using the Recovery RL algorithm [36] 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 5 (**LS^3 ($-$Safe Set)**) to evaluate if the safe set promotes consistent task completion and stable learning. Finally, we compare LS^3 to a an ablated version of the safe set classifier (Section 4.2) without a recursive objective, where the classifier is just trained to predict $\mathbb{1}_{\mathbb{S}^J}$ (**LS^3 (BC SS)**). See the supplement for details on hyperparameters and offline data used for LS^3 and prior algorithms.

5.2 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.

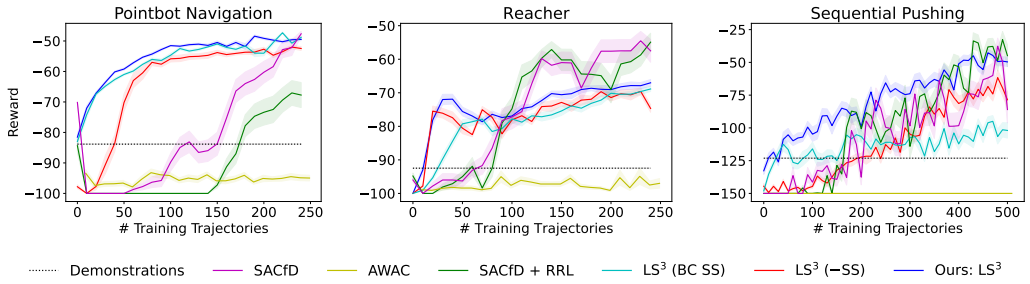

Figure 4: **Simulation Experiments Results:** Learning curves showing mean and standard error over 10 random seeds. We see that LS^3 learns more quickly than baselines and ablations. Although SACfD and SACfD+RRL converge to similar reward values, LS^3 is much more sample efficient and stable across random seeds.

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, 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.

5.3 Domains

In simulation, we evaluate LS^3 on 3 vision-based continuous control domains that are illustrated in Figure 3. 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 2. We then study a constrained image-based reaching task (Reacher) based on [50], 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) [51] (Figure 3), 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. See the supplement for more details on all domains.

5.4 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 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 1 and Table 2, 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 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.

Table 1: **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 2: **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 off 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

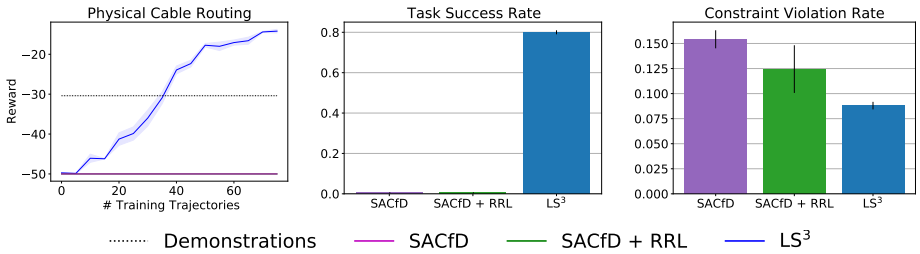

Figure 5: **Physical Cable Routing Results:** We present learning curves, task success rates and constraint violation rates with a mean and standard error across 3 random seeds. LS^3 learns a more efficient policy than the demonstrator while still violating constraints less than comparisons, which are unable to learn the task.

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 4.2 (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.

5.5 Physical Results

In physical experiments, we compare LS^3 to SACfD and SACfD+RRL (Figure 5) on the physical cable routing task illustrated in Figure 3. 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.

6 Discussion and Future Work

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. In future work, we are excited to explore further physical evaluation of LS^3 on safety critical visuomotor control tasks and applications to systems with dynamic constraints on velocity or acceleration.

References

[1] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[2] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *Proc. Int. Conf. on Machine Learning*, 2019.

[3] R. Hoque, D. Seita, A. Balakrishna, A. Ganapathi, A. K. Tanwani, N. Jamali, K. Yamane, S. Iba, and K. Goldberg. Visuospatial foresight for multi-step, multi-task fabric manipulation. *Proc. Robotics: Science and Systems (RSS)*, 2020.

[4] I. Lenz, R. A. Knepper, and A. Saxena. Deepmpc: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*. Rome, Italy, 2015.

[5] S. Nair and C. Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. *Proc. Int. Conf. on Learning Representations*, 2019.

[6] S. Nair, S. Savarese, and C. Finn. Goal-aware prediction: Learning to model what matters. In *Proceedings of the 37th International Conference on Machine Learning*, pages 7207–7219, 2020.

[7] K. Pertsch, O. Rybkin, F. Ebert, C. Finn, D. Jayaraman, and S. Levine. Long-horizon visual planning with goal-conditioned hierarchical predictors. *Proc. Advances in Neural Information Processing Systems*, 2020.

[8] S. Tian, S. Nair, F. Ebert, S. Dasari, B. Eysenbach, C. Finn, and S. Levine. Model-based visual planning with self-supervised functional distances. *Proc. Int. Conf. on Learning Representations*, 2021.

[9] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications.

[10] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. *Proc. Advances in Neural Information Processing Systems*, 2018.

[11] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.

[12] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Conf. on Robot Learning (CoRL)*, 2018.

[13] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine. Skew-fit: State-covering self-supervised reinforcement learning. *Proc. Int. Conf. on Machine Learning*, 2020.

[14] U. Rosolia and F. Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 2018.

[15] S. M. Richards, F. Berkenkamp, and A. Krause. The lyapunov neural network: Adaptive stability certification for safe learning of dynamical systems. In *Conference on Robot Learning*, pages 466–476. PMLR, 2018.

[16] B. Thananjeyan, A. Balakrishna, U. Rosolia, J. E. Gonzalez, A. Ames, and K. Goldberg. Abc-lmpc: Safe sample-based learning mpc for stochastic nonlinear dynamical systems with adjustable boundary conditions, 2020.

[17] B. Thananjeyan, A. Balakrishna, U. Rosolia, F. Li, R. McAllister, J. E. Gonzalez, S. Levine, F. Borrelli, and K. Goldberg. Safety augmented value estimation from demonstrations (saved): Safe deep model-based rl for sparse cost robotic tasks. *IEEE Robotics and Automation Letters*, 5(2):3612–3619, 2020.

[18] U. Rosolia and F. Borrelli. Sample-based learning model predictive control for linear uncertain systems. *CoRR*, abs/1904.06432, 2019. URL http://arxiv.org/abs/1904.06432.

[19] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin. Hamilton-jacobi reachability: A brief overview and recent advances. In *Conference on Decision and Control (CDC)*, 2017.

[20] D. A. Bristow, M. Tharayil, and A. G. Alleyne. A survey of iterative learning control. *IEEE control systems magazine*, 2006.

[21] U. Rosolia, X. Zhang, and F. Borrelli. A Stochastic MPC Approach with Application to Iterative Learning. *2018 IEEE Conference on Decision and Control (CDC)*, 2018.

[22] M. Deisenroth and C. Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proc. Int. Conf. on Machine Learning*, 2011.

[23] I. Lenz, R. A. Knepper, and A. Saxena. DeepMPC: Learning deep latent features for model predictive control. In *Robotics: Science and Systems*, 2015.

[24] J. Fu, S. Levine, and P. 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.

[25] K. Lowrey, A. Rajeswaran, S. Kakade, E. Todorov, and I. Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In *Proc. Int. Conf. on Machine Learning*, 2019.

[26] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proc. Advances in Neural Information Processing Systems*, 2018.

[27] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2018.

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

[29] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg. Deep learning can accelerate grasp-optimized motion planning. *Science Robotics*, 5(48), 2020.

[30] Z. Zhu, N. Pivaroa, S. Gupta, A. Gupta, and M. Canova. 2021.

[31] M. Lippi, P. Poklukar, M. C. Welle, A. Varava, H. Yin, A. Marino, and D. Kragic. Latent space roadmap for visual action planning of deformable and rigid object manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.

[32] A. A. Rusu, M. Večerík, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning*, pages 262–270. PMLR, 2017.

[33] G. Schoettler, A. Nair, J. Luo, S. Bahl, J. A. Ojea, E. Solowjow, and S. Levine. Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2020.

[34] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-end robotic reinforcement learning without reward engineering. *Proc. Robotics: Science and Systems (RSS)*, 2019.

[35] M. Zhang, S. Vikram, L. Smith, P. Abbeel, M. Johnson, and S. Levine. Solar: Deep structured representations for model-based reinforcement learning. In *International Conference on Machine Learning*, pages 7444–7453. PMLR, 2019.

[36] B. Thananjeyan, A. Balakrishna, S. Nair, M. Luo, K. Srinivasan, M. Hwang, J. E. Gonzalez, J. Ibarz, C. Finn, and K. Goldberg. Recovery rl: Safe reinforcement learning with learned recovery zones. *NeurIPS Deep Reinforcement Learning Workshop*, 2020.

[37] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine. Uncertainty-aware reinforcement learning for collision avoidance. *CoRR*, 2017.

[38] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. Universal planning networks. *Proc. Int. Conf. on Machine Learning*, 04 2018.

[39] B. Ichter and M. Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019. doi:10.1109/LRA.2019.2901898.

[40] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *Proc. Int. Conf. on Learning Representations*, 2017.

[41] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. 2020. arXiv:2004.14990.

[42] R. Rubinstein. The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*, 1(2):127–190, 1999.

[43] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proc. Advances in Neural Information Processing Systems*, 2018.

[44] J. Zhang, B. Cheung, C. Finn, S. Levine, and D. Jayaraman. Cautious adaptation for reinforcement learning in safety-critical settings. In *International Conference on Machine Learning*, pages 11055–11065. PMLR, 2020.

[45] P. Kazanzides, Z. Chen, A. Deguet, G. S. Fischer, R. H. Taylor, and S. P. DiMaio. An open-source research kit for the da Vinci surgical system. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2014.

[46] D. Seita, S. Krishnan, R. Fox, S. McKinley, J. Canny, and K. Goldberg. 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.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *Proc. Int. Conf. on Machine Learning*, 2018.

[48] A. Nair, A. Gupta, M. Dalal, and S. Levine. Awac: Accelerating online reinforcement learning with offline datasets, 2021.

[49] C. Tessler, D. J. Mankowitz, and S. Mannor. Reward constrained policy optimization. In *Proc. Int. Conf. on Learning Representations*, 2019.

[50] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dm-control: Software and tasks for continuous control, 2020.

[51] P. Kazanzides, Z. Chen, A. Deguet, G. S. Fischer, R. H. Taylor, and S. P. DiMaio. An open-source research kit for the da vinci® surgical system. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 6434–6439. IEEE, 2014.

[52] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. Int. Conf. on Machine Learning*.

[53] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. 37:1889–1897, 07–09 Jul 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

[54] K. Chua. Experiment code for "deep reinforcement learning in a handful of trials using probabilistic dynamics models". https://github.com/kchua/handful-of-trials, 2018.

[55] V. Pong. rlkit. https://github.com/vitchyr/rlkit, 2018.

[56] B. T. Ashwin Balakrishna. Code for recovery rl. https://github.com/abalakrishna123/recovery-rl, 2021.

[57] H. Sikchi. Code for advantage weighted actor critic. https://github.com/hari-sikchi/AWAC, 2021.

[58] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012. URL https://ieeexplore.ieee.org/abstract/document/6386109/.

7 Appendix

In Appendices 7.1 and 7.2 we discuss algorithmic details and implementation/hyperparameter details respectively for LS^3 and all comparisons. We then provide full details regarding each of the experimental domains and how data is collected in these domains in Appendix 7.3. In Appendix 7.4, we provide results from an additional experiment studying the task success rate of LS^3 and comparisons evolves as training progresses. Finally, in Appendix 7.5 we perform sensitivity experiments and ablations.

7.1 Algorithm Details

In this section, we provide implementation details and additional background information for LS^3 and comparison algorithms.

7.1.1 Latent Space Safe Sets (LS^3)

We now discuss additional details for each of the components of LS^3, including network architectures, training data, and loss functions.

Variational Autoencoders: 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 Hafner et al. [2], 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

$$J(\theta) = \|f_{\text{dec}}(z_t) - s_t\|_2^2 + \beta D_{KL}\left(f_{\text{enc}}(z_t|s_t)||\mathcal{N}(0,1)\right) \tag{7}$$

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

Probabilistic Dynamics: As in Chua et al. [43] 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:

$$J(\theta) = -\log f_{\text{dyn},\theta}(z_{t+1}|z_t, a_t) \tag{8}$$

When using the learned dynamics model f_{dyn} for planning, we use the TS-1 method from Chua et al. [43].

Value Functions: As discussed in Section 4.3, 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 [17], 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:

$$J(\theta) = \left(V_\theta^\pi(z_t) - \sum_{i=1}^{T-t} \gamma^i r_{t+i}\right)^2 \tag{9}$$

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

$$J(\theta) = \left(V_\theta^\pi(z_t) - (r_t + \gamma V_{\theta'}^{\pi'}(z_{t+1}))\right)^2 \tag{10}$$

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.

Constraint and Goal Estimators: We represent constraint indicator $f_{\mathcal{C}} : \mathcal{Z} \to \{0, 1\}$ 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 $f_{\mathcal{G}} : \mathcal{Z} \to \{0, 1\}$ 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 $\mathcal{D}_{\text{constraint}}$.

Safe Set: 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

$$f_{\mathbb{S}}(s_t) = \max(\mathbb{1}_{\mathbb{S}^j}(s_t), \gamma_{\mathbb{S}} f_{\mathbb{S}}(s_{t+1})) \tag{11}$$

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 [52, 53, 17], 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.

Cross Entropy Method: We use the cross entropy method to solve the optimization problem in equation 2. We build on the implementation of the cross entropy method provided in [54], 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 2 where constraints are implemented by assigning large negative rewards to trajectories which violate either the safe 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 2. 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 Chua et al. [43] for more details on the cross entropy method as applied to planning over neural network dynamics models.

As mentioned in Section 4.4, 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 $\delta_{\mathbb{S}}$. This ensures that $\delta_{\mathbb{S}}$ is set such that planning back to the safe set is possible. We initialize $\delta_{\mathbb{S}} = 0.8$.

7.1.2 Soft Actor-Critic from Demonstrations (SACfD)

We utilize the implementation of the Soft Actor Critic algorithm from [55] and initialize the actor and critic from demonstrations but keep all other hyperparameters the same as the default in the provided implementation. We create a new dataset $\mathcal{D}_{\text{demos}} \subsetneq \mathcal{D}$ using only data from the suboptimal demonstrator, and use the data from $\mathcal{D}_{\text{demos}}$ to behavior clone the actor and initialize the critic using offline bellman backups. We use the same mean-squared loss function for behavior cloning as for the behavior clone policy, but only train the mean of the SAC policy. Precisely, we use the following loss for some policy π with parameter θ: $L(\theta, \mathcal{D}_{\text{demos}}) = \sum_{\tau_i \in \mathcal{D}_{\text{demos}}} \sum_{t=1}^{T} ||\mu_\theta(s_t^i) - a_t^i||^2$ where s_t^i and a_t^i are the state and action at timestep t of trajectory τ_i and $\pi(\cdot|s_t) \sim \mathcal{N}(\mu_\theta(s_t), \sigma_\phi(s_t))$. We also experimented with training the SAC critic on all data provided to LS^3 in $\mathcal{D}$ but found that this hurt performance. We use the architecture from [55] and update neural network weights using an Adam optimizer with a learning rate of 3×10^{-4}. The only hyperparameter for SACfD that we tuned across environments was the reward penalty λ which was imposed upon constraint violations. For all simulation experiments, we evaluated $\lambda \in \{-1, -3, -5, -10, -20\}$ and report the highest performing value. Accordingly, we use $\lambda = -3$ for all experiments except the reacher task, for which we used $\lambda = -1$. We observed that higher values of λ resulted in worse task performance without significant increase in constraint satisfaction. We hypothesize that since the agent is frozen in the environment upon constraint violations, the resulting loss of rewards from this is sufficient to enable SACfD to avoid constraint violations.

Table 3: 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

7.1.3 Soft Actor-Critic from Demonstrations with Learned Recovery Zones (SACfD+RRL)

We build on the implementation of the Recovery RL algorithm [36] provided in [56]. We train the safety critic on all offline data from $\mathcal{D}$. Recovery RL uses SACfD as its task policy optimization algorithm, and introduces two new hyperparameters: $(\gamma_{\text{risk}}, \epsilon_{\text{risk}})$. For each of the simulation environments, we evaluated SACfD+RRL across 3-4 $(\gamma_{\text{risk}}, \epsilon_{\text{risk}})$ settings and reported results from the highest performing run. Accordingly, for the navigation environment, we use: $(\gamma_{\text{risk}} = 0.95, \epsilon_{\text{risk}} = 0.8)$. For the reacher environment, we use $(\gamma_{\text{risk}} = 0.55, \epsilon_{\text{risk}} = 0.7)$, and we use $(\gamma_{\text{risk}} = 0.75, \epsilon_{\text{risk}} = 0.7)$ for the sequential pushing environment. For the cable routing environment, we use $(\gamma_{\text{risk}} = 0.55, \epsilon_{\text{risk}} = 0.7)$.

7.1.4 Advantage Weighted Actor-Critic (AWAC)

To provide a comparison to state of the art offline reinforcement learning algorithms, we evaluate AWAC [48] on the experimental domains in this work. We use the implementation of AWAC from [57]. For all simulation experiments, we evaluated $\lambda \in \{-1, -3, -5, -10, -20\}$ and report the highest performing value. Accordingly, we use $\lambda = -1$ for all experiments. We used the default settings from [57] for all other hyperparameters.

7.2 LS^3 Implementation Details

In Table 3, 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 $\gamma_{\mathbb{S}}$. 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}.

7.3 Experimental Domain Details

7.3.1 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.

7.3.2 Reacher

The reacher domain is built on the reacher domain provided in the DeepMind Control Suite from [50]. 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.

7.3.3 Sequential Pushing

This sequential pushing environment is implemented in MuJoCo [58], 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.

7.3.4 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 $|\mathcal{D}_{\text{VAE}}| > 100000$.

7.4 Additional Results

We additionally study how the task success rate of LS^3 and comparisons evolves as training progresses in Figure 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.

7.5 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 7 and 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 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.

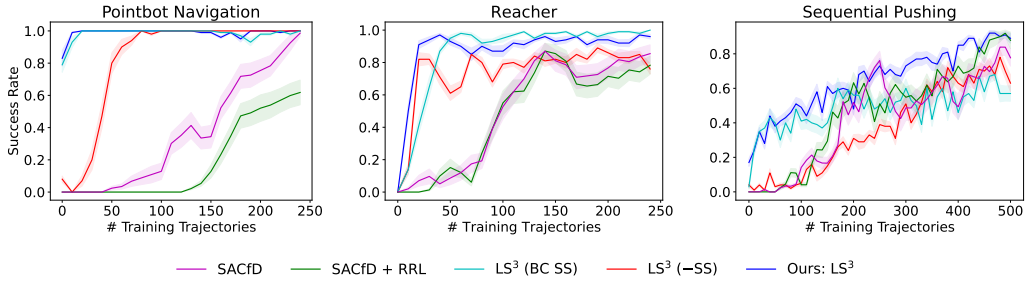

Figure 6: **Task Success Rate:** Learning curves showing mean and standard error of task success rate of checkpointed policies over 10 random seeds (and 10 rollouts per seed). We see that LS^3 has a much higher task success rate than comparisons early on, and maintains a success rate at least as high as comparisons throughout training in all environments.

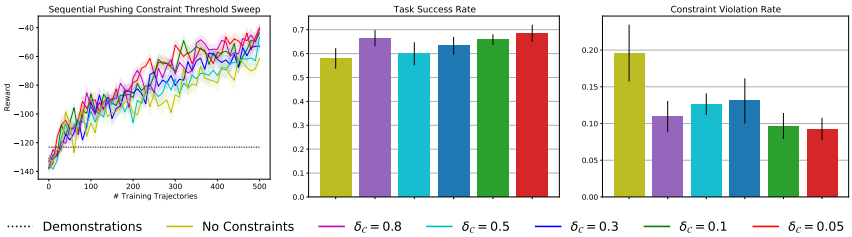

Figure 7: **Hyperparameter Sweep for LS^3 Constraint Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathcal{C}}$ on the sequential pushing environment. As expected, we see that without avoiding latent space obstacles (No Constraints) the agent violates constraints more often, while lower thresholds (meaning the planning algorithm is more conservative) generally lead to fewer violations.

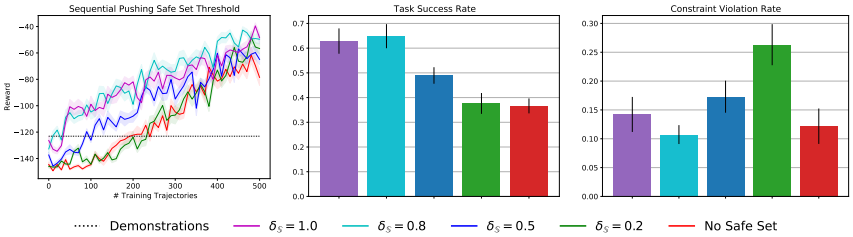

Figure 8: **Hyperparameter Sweep for LS^3 Safe Set Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathbb{S}}$ on the sequential pushing environment. We see that after offline training, the agent can successfully complete the task 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.

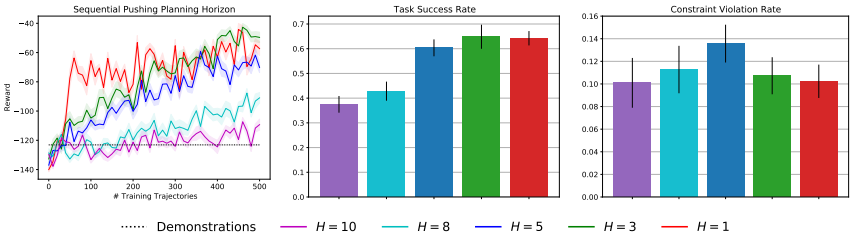

Figure 9: **Hyperparameter Sweep for LS^3 Planning Horizon:** Plots show mean and standard error over 10 random seeds for experiments with different settings of H on the sequential pushing environment. We see that when the planning horizon is too high the agent cannot reliably complete the task due to 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.