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

Albert Wilcox*, **Ashwin Balakrishna***, **Brijen Thananjeyan**,
Joseph E. Gonzalez[1], **Ken Goldberg**[1]
* equal contribution
{albertwilcox, ashwin_balakrishna}@berkeley.edu

1 Appendix

In Appendices 1.1 and 1.2 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 1.3. In Appendix 1.4, we present an additional experiment studying the task success rate of LS^3 and comparisons evolves as training progresses. Finally, in Appendix 1.5 we perform sensitivity experiments and ablations.

1.1 Algorithm Details

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

1.1.1 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. [1], 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

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

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

Probabilistic Dynamics: As in Chua et al. [2] 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:

$$J(\theta) = -\log f_{\text{dyn},\theta}(z_{t+1}|z_t, a_t) \tag{2}$$

When using f_{dyn} for planning, we use the TS-1 method from Chua et al. [2].

Value Functions: As discussed in Section **??**, 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 [3], we use separate training objectives during offline and online training. During offline training, we train the function to predict

University of California, Berkeley.

5th Conference on Robot Learning (CoRL 2021), London, UK.

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:

$$J(\theta) = \left(V_\theta^\pi(z_t) - \sum_{i=1}^{T-t} \gamma^i r_{t+i} \right)^2 \tag{3}$$

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

$$J(\theta) = \left(V_\theta^\pi(z_t) - (r_t + \gamma V_{\theta'}^{\pi'}(z_{t+1})) \right)^2 \tag{4}$$

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

$$f_{\mathbb{S}}(s_t) = \max(\mathbb{1}_{\mathbb{S}^j}(s_t), \gamma_{\mathbb{S}} f_{\mathbb{S}}(s_{t+1})) \tag{5}$$

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 [4, 5, 3], 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 **??**. 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 **??** 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 **??**. 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. [2] for more details on the cross entropy method as applied to planning over neural network dynamics models.

As mentioned in Section **??**, 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 $\delta_{\mathbb{S}} = 0.8$.

1.1.2 Soft Actor-Critic from Demonstrations (SACfD)

We utilize the implementation of the Soft Actor Critic algorithm from [7] and initialize the actor and critic from demonstrations but keep all other hyperparameters the same as the default in the

Table 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

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 [7] 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.

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

We build on the implementation of the Recovery RL algorithm [8] provided in [9]. 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 $(\gamma_{\text{risk}} = 0.55, \epsilon_{\text{risk}} = 0.7)$.

1.1.4 Advantage Weighted Actor-Critic (AWAC)

To provide a comparison to state of the art offline reinforcement learning algorithms, we evaluate AWAC [10] on the experimental domains in this work. We use the implementation of AWAC from [11]. 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 [11] for all other hyperparameters.

1.2 LS^3 Implementation Details

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

1.3 Experimental Domain Details

1.3.1 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.

1.3.2 Reacher

The reacher domain is built on the reacher domain provided in the DeepMind Control Suite from [12]. 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.

1.3.3 Sequential Pushing

This sequential pushing environment is implemented in MuJoCo [13], 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.

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

1.4 Additional Results

We additionally study how the task success rate of LS^3 and comparisons evolves as training progresses in Figure 1. 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.

1.5 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 2 and 4. 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.

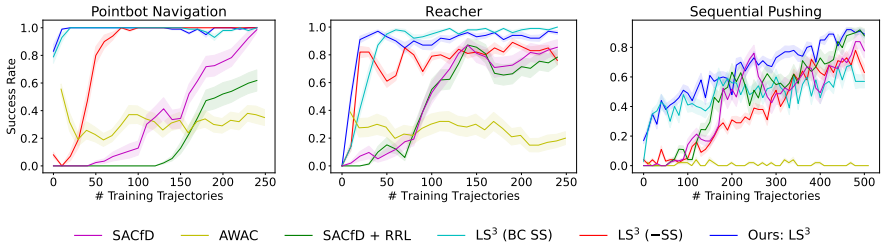

Figure 1: **Task Success Rate:** Learning curves showing mean and standard error of task success rate of checkpointed policies over 10 random seeds (and 10 rollouts per seed). We see that LS^3 has a much higher task success rate than comparisons early on, and maintains a success rate at least as high as comparisons throughout training in all environments.

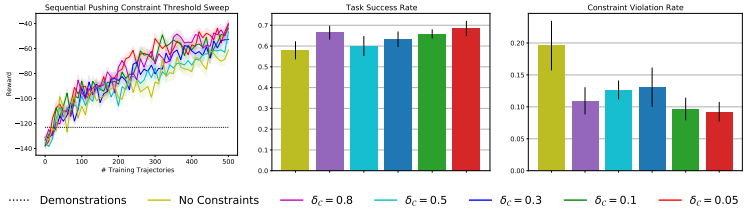

Figure 2: **Hyperparameter Sweep for LS^3 Constraint Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathcal{C}}$ on the sequential pushing environment. As expected, we see that without avoiding latent space obstacles (No Constraints) the agent violates constraints more often, while lower thresholds (meaning the planning algorithm is more conservative) generally lead to fewer violations.

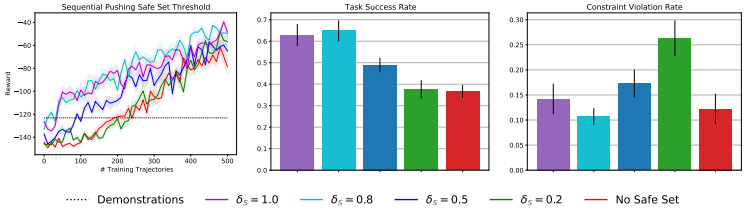

Figure 3: **Hyperparameter Sweep for LS^3 Safe Set Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathbb{S}}$ on the sequential pushing environment. We see that after offline training, the agent can successfully complete the task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

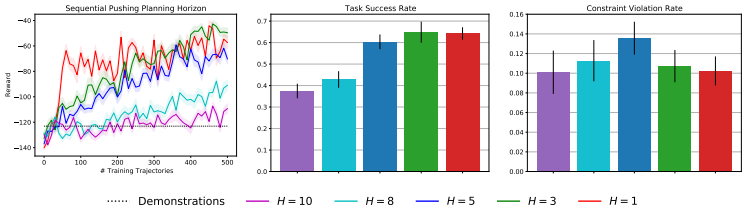

Figure 4: **Hyperparameter Sweep for LS^3 Planning Horizon:** Plots show mean and standard error over 10 random seeds for experiments with different settings of H on the sequential pushing environment. We see that when the planning horizon is too high the agent cannot reliably complete the task due to 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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*.

[5] 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.

[6] 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.

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

[8] 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.

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

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

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

[12] 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.

[13] 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/.