Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones

Brijen Thananjeyan[*1], **Ashwin Balakrishna**[*1], **Suraj Nair**[2], **Michael Luo**[1], **Krishnan Srinivasan**[2], **Minho Hwang**[1], **Joseph E. Gonzalez**[1], **Julian Ibarz**[3], **Chelsea Finn**[2], **Ken Goldberg**[1]

* equal contribution

{bthananjeyan, ashwin_balakrishna}@berkeley.edu

Abstract: Reinforcement learning (RL) provides a flexible and general-purpose framework for learning new behaviors through interaction with the environment. However, safety remains a central obstacle preventing widespread use of RL algorithms in the real world since learning new tasks in unknown environments requires extensive exploration, but safety requires limiting exploration. We propose Recovery RL, an algorithm which navigates this tradeoff by (1) efficiently leveraging offline data to learn about constraint violating zones *before* policy learning and (2) *separating* the goals of improving task performance and constraint satisfaction across two policies: a task policy that only optimizes the task reward and a recovery policy that guides the agent back to safety when constraint violation is likely. This recovery mechanism can be applied in conjunction with any RL algorithm, and essentially defines a new MDP where exploration is probabilistically safe for the task policy. We evaluate Recovery RL on 6 continuous control domains in simulation, including two contact rich manipulation tasks and an image-based navigation task and compare Recovery RL to 5 prior safe RL methods which jointly optimize task performance and safety via constrained optimization or reward shaping. Results suggest that Recovery RL trades off constraint violations and task successes 2 - 80 times more efficiently than the next best prior method across simulation domains. We then evaluate Recovery RL on an image-based constrained reaching task on a physical robot and find that Recovery RL trades off constraint violations and task successes 12 times more efficiently than the next best prior algorithm.

Keywords: Reinforcement Learning, Imitation Learning, Safety

1 Introduction

Reinforcement learning (RL) provides a general framework for robots to acquire new skills, and has shown promise in a variety of robotic domains such as navigation [1], locomotion [2], and manipulation [3, 4]. However, enforcing constraints on the agent's behavior to encourage safety during learning and exploration is challenging, since constraint violating states and the states leading to them may be initially unknown and must be learned from experience. Thus, safe exploration requires navigating a tradeoff: learning new skills through environmental interaction requires exploring a wide range of possible behaviors, but learning safely forces the agent to restrict exploration to constraint satisfying states. For example, if an agent is tasked to navigate between two ends of a tight passage, the task reward will encourage an RL policy to enter the passage, while minimizing the probability of constraint violations will encourage the policy to avoid the passage entirely.

We consider a RL formulation subject to constraints on the probability of unsafe future behavior and design an algorithm that can effectively balance the often conflicting objectives of task directed exploration and safety. Most prior work in safe RL integrates constraint satisfaction into the task objective to jointly optimize the two. While these approaches are appealing for their generality and simplicity, there are two key aspects which make them difficult to apply in practice. First, the inherent objective conflict between exploring sufficiently to learn a good task policy and limiting exploration to avoid constraint violations can lead to suboptimalities in policy optimization. Second, sufficiently

[1]University of California, Berkeley. [2]Stanford University. [3]Google Brain Robotics.

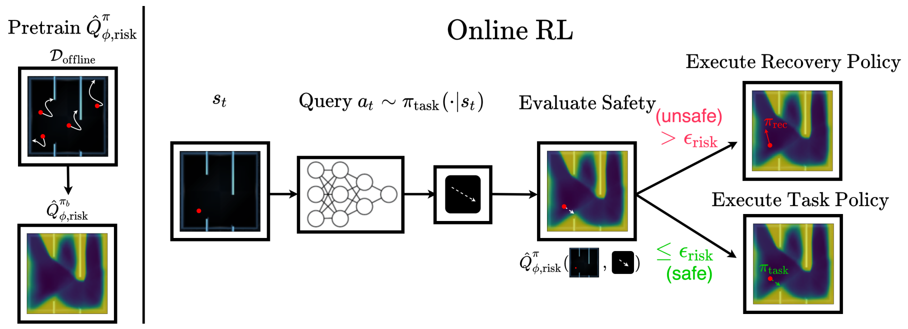
Pretrain $\hat{Q}^{\pi}_{\phi,\text{risk}}$
$\mathcal{D}_{\text{offline}}$
$\hat{Q}^{\pi_b}_{\phi,\text{risk}}$
Online RL
s_t
Query $a_t \sim \pi_{\text{task}}(\cdot|s_t)$
Evaluate Safety
$\hat{Q}^{\pi}_{\phi,\text{risk}}(\ ,\ )$
(unsafe)
$> \epsilon_{\text{risk}}$
$\leq \epsilon_{\text{risk}}$
(safe)
Execute Recovery Policy
π_{rec}
Execute Task Policy
π_{task}

Figure 1: **Recovery RL:** We illustrate Recovery RL on a 2D maze navigation task where a constraint violation corresponds to hitting a wall. Recovery RL first learns safety critic $\hat{Q}^{\pi}_{\phi,\text{risk}}$ with offline data from some behavioral policy π_b, which provides a small number of controlled demonstrations of constraint violating behavior as shown on the left. For the purposes of illustration, we visualize the average of the $\hat{Q}^{\pi}_{\phi,\text{risk}}$ learned by Recovery RL over 100 action samples. Then, at each timestep, Recovery RL queries the task policy π_{task} for some action a at state s, evaluates $\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a)$, and executes the recovery policy π_{rec} if $\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a) > \epsilon_{\text{risk}}$ and π_{task} otherwise. The task policy, recovery policy, and safety critic are updated after each transition from agent experience.

exploring the environment to learn about constraint structure necessitates a significant amount of constraint violations during learning. However, this can result in the agent taking uncontrolled actions which can damage both itself and the environment.

We take a step towards addressing these issues with two key algorithmic ideas. First, inspired by recent work in robust control [5, 6, 7, 8], we represent the RL agent with two policies: the first policy focuses on optimizing the unconstrained task objective (task policy) and the second policy takes control when the task policy is in danger of constraint violations in the near future (recovery policy). Instead of modifying the policy optimization procedure to encourage constraint satisfaction, which can introduce suboptimality in the learned task policy [9], the recovery policy can be viewed as defining an alternate MDP for the task policy to explore within in which constraint violations are unlikely. Separating the task policy and the recovery policy makes it easier to balance task performance and safety, and allows us to apply off-the-shelf RL algorithms for learning each. Second, we leverage offline data to learn a recovery set, which indicates regions of the MDP in which future constraint violations are likely, and a recovery policy, which is queried within this set to prevent violations. This offline data can be collected by a human or an agent under human supervision to provide controlled examples of constraint violations, such as gently tipping over a glass rather than aggressively knocking the glass over and shattering it. Thus, the agent is able to observe constraint violations and learn from them without the task policy directly having to experience too many uncontrolled examples of these violations during learning.

We present Recovery RL, a new algorithm for safe robotic RL. Unlike prior work, Recovery RL (1) can effectively leverage offline data of constraint violations to learn about constraints *before* interacting with the environment, and (2) uses separate policies for the task and recovery to learn safely without significantly sacrificing task performance. We evaluate Recovery RL against 5 state-of-the-art safe RL algorithms on 6 navigation and manipulation domains in simulation, including a visual navigation task, and find that Recovery RL trades off constraint violations and task successes 2 - 80 times more efficiently than the next best prior method. We then evaluate Recovery RL on a constrained image-based reaching task on a physical robot and find that Recovery RL trades off constraint violations and task successes 12 times more efficiently than the next best prior algorithm.

2 Related Work

Prior work has studied safety in RL in several ways, including imposing constraints on expected return [10, 11], risk measures [12, 13, 14, 15], and avoiding regions of the MDP where constraint violations are likely [16, 17, 6, 18, 19, 20]. We build on the latter approach, and design an algorithm which uses a learned recovery policy to keep the RL agent within a learned safe region of the MDP.

Jointly Optimizing for Task Performance and Constraint Satisfaction: A popular strategy in algorithms for safe RL involves modifying the policy optimization procedure of standard RL algorithms to simultaneously reason about both task reward and constraints using methods such as trust regions [10], optimizing a Lagrangian relaxation [11, 21, 22], or constructing Lyapunov functions [23, 24]. The most similar of these works to Recovery RL is [22]. Srinivasan et al. [22] trains a safety critic, which estimates the probability of future constraint violation under the current policy, and optimizes a Lagrangian objective function to limit the probability of constraint violations while maximizing task reward. Unlike Srinivasan et al. [22], which uses the safety critic to modify the task policy optimization objective, Recovery RL uses it to determine when to execute a learned recovery policy which minimizes the safety critic to keep the agent in safe regions of the MDP. This idea enables Recovery RL to more effectively balance task performance and constraint satisfaction than algorithms which jointly optimize for task performance and safety.

Restricting Exploration with an Auxiliary Policy: Another approach to safe RL explicitly restricts policy exploration to a safe subset of the MDP using a recovery or shielding mechanism. This idea has been explored in [5, 6], which utilize Hamilton-Jacobi reachability analysis to define a task policy and safety controller, and in the context of shielding [7, 8, 25]. In contrast to these works, which assume approximate knowledge of system dynamics [5, 6, 7, 8, 25] or require precise knowledge of constraints apriori [25], Recovery RL learns information about the MDP, such as constraints and dynamics, from experience and can scale to high-dimensional state spaces. Additionally, Recovery RL reasons about probabilistic constraints rather than robust constraints, allowing it to estimate a safe set without a dynamics model. Han et al. [26] and Eysenbach et al. [16] introduce reset policies which are trained jointly with the task policy to reset the agent to its initial state distribution, ensuring that the task policy only learns behaviors which can be reset [16]. However, enforcing the ability to fully reset can be impractical or inefficient. Inspired by this work, Recovery RL instead executes approximate resets to nearby safe states when constraint violation is probable. Similar to Recovery RL, Richter and Roy [1] learns the probability of constraint violation conditioned on an action plan to activate a hand-designed safety controller. In contrast, Recovery RL uses a learned recovery mechanism which can be broadly applied across different tasks.

Leveraging Demonstrations for Safe RL and Control: Finally, there has also been significant prior work investigating how demonstrations can be leveraged to enable safe exploration. Rosolia and Borrelli [27], Thananjeyan et al. [28] introduce model predictive control algorithms which leverage initial constraint satisfying demonstrations to iteratively improve their performance with safety guarantees and Thananjeyan et al. [18] extends these ideas to the RL setting. In contrast to these works, Recovery RL learns a larger safe set that explicitly models future constraint satisfaction and also learns the problem constraints from prior experience without task specific demonstrations. Additionally, Recovery RL can be applied with either model-free or model-based RL algorithms while [18, 28] require a dynamics model to evaluate reachability-based safety online.

3 Problem Statement

We consider RL under Markov decision processes (MDPs), which can be described by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P(\cdot|\cdot,\cdot), R(\cdot,\cdot), \gamma, \mu)$ where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces. The stochastic dynamics model $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ maps a state and action to a probability distribution over subsequent states, $\gamma \in [0, 1]$ is a discount factor, μ is the initial state distribution ($s_0 \sim \mu$), and $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. We augment the MDP with an additional constraint cost function $C : \mathcal{S} \to \{0, 1\}$ which indicates whether a state is constraint violating and an associated discount factor $\gamma_{\text{risk}} \in [0, 1]$. This yields the following new MDP: $(\mathcal{S}, \mathcal{A}, P(\cdot|\cdot,\cdot), R(\cdot,\cdot), \gamma, C(\cdot), \gamma_{\text{risk}})$. We assume that episodes terminate on constraint violation, which is equivalent to transitioning to a constraint-satisfying absorbing state with zero reward.

Let Π be the set of Markovian stationary policies. Given policy $\pi \in \Pi$, the expected return is defined as $R^\pi = \mathbb{E}_{\pi,\mu,P}\left[\sum_t \gamma^t R(s_t, a_t)\right]$ and the expected discounted probability of constraint violation is defined as $Q^\pi_{\text{risk}}(s_i, a_i) = \mathbb{E}_{\pi,\mu,P}\left[\sum_t \gamma^t_{\text{risk}} C(s_{t+i})\right] = \sum_t \gamma^t_{\text{risk}} \mathbb{P}\left(C(s_{t+i}) = 1\right)$, which we would like to be below a threshold $\epsilon_{\text{risk}} \in [0, 1]$. The objective of Recovery RL is to solve the following constrained optimization problem:

$$\pi^* = \arg\max_{\pi \in \Pi} \{R^\pi : Q^\pi_{\text{risk}}(s_0, a_0) \le \epsilon_{\text{risk}}\} \tag{1}$$

This setting exactly corresponds to the CMDP formulation from [29], but with constraint costs limited to binary indicator functions for constraint violating states. We limit the choice to binary indicator functions, as they are easier to provide than shaped costs and use Q^π_{risk} to convey information about delayed constraint costs. We define the set of feasible policies, $\{\pi : Q^\pi_{\text{risk}} \le \epsilon\}$, the set of ϵ-safe policies Π_ϵ. Observe that if $\gamma_{\text{risk}} = 1$, then by the assumption of termination on constraint violation,

$Q^{\pi}_{\text{risk}}(s_i, a_i) = \mathbb{P}\left(\bigcup_t C(s_t) = 1\right)$, or the probability of a constraint violation in the future. Setting $\epsilon_{\text{risk}} = 0$ as well results in a robust optimal control problem.

We present an algorithm to optimize equation (1) by utilizing a pair of policies, a *task policy* π_{task}, which is trained to maximize R^{π} over $\pi_{\text{task}} \in \Pi$ and a *recovery policy* π_{rec}, which attempts to guide the agent back to a state-action tuple (s, a) where $Q^{\pi}_{\text{risk}}(s, a) \leq \epsilon_{\text{risk}}$. We additionally assume access to a set of transitions from offline data ($\mathcal{D}_{\text{offline}}$) that contains examples of constraint violations. Unlike demonstrations in typical imitation learning settings, this data need not illustrate task successes, but rather shows possible ways to violate constraints. We leverage $\mathcal{D}_{\text{offline}}$ to constrain exploration of the task policy to reduce the probability of constraint violation during environment interaction.

4 Recovery RL

Here we outline the central ideas behind Recovery RL. In Section 4.1, we review how to learn a safety critic to estimate the probability of future constraint violations. Then in Section 4.2, we show how this safety critic can be used to define the recovery policy for Recovery RL and the recovery set in which it is activated. In Section 4.3 we discuss how the safety critic and recovery policy are learned from offline data and in Section 4.4 we discuss implementation details.

4.1 Preliminaries: Training a Safety Critic

As in Srinivasan et al. [22], Recovery RL learns a critic function Q^{π}_{risk} that estimates the discounted future probability of constraint violation of the policy π being executed currently in the environment:

$$\begin{aligned} Q^{\pi}_{\text{risk}}(s_t, a_t) &= \mathbb{E}_{\pi}\left[\sum_{t'=t}^{\infty} \gamma_{\text{risk}}^{t'-t} C(s_{t'}) | s_t, a_t\right] \\ &= C(s_t) + (1 - C(s_t))\gamma_{\text{risk}}\mathbb{E}_{\pi}\left[Q^{\pi}_{\text{risk}}(s_{t+1}, a_{t+1}) | s_t, a_t\right]. \end{aligned} \tag{2}$$

This is different from the standard Bellman equations for solving MDPs due to the assumption that episodes terminate when $C(s_t) = 1$. In practice, we train a sample-based approximation $\hat{Q}^{\pi}_{\phi,\text{risk}}$, parameterized by ϕ, by approximating these equations using sampled transitions $(s_t, a_t, s_{t+1}, C(s_t))$.

We train $\hat{Q}^{\pi}_{\phi,\text{risk}}$ by minimizing the following MSE loss with respect to the target (RHS of equation 2).

$$\begin{aligned} &J_{\text{risk}}(s_t, a_t, s_{t+1}; \phi) = \\ &\frac{1}{2}\left(\hat{Q}^{\pi}_{\phi,\text{risk}}(s_t, a_t) - (C(s_t) + (1 - C(s_t))\gamma_{\text{risk}} \mathop{\mathbb{E}}_{a_{t+1}\sim\pi(\cdot|s_{t+1})}[\hat{Q}^{\pi}_{\phi,\text{risk}}(s_{t+1}, a_{t+1})])\right)^2 \end{aligned} \tag{3}$$

In practice, we use a target network to create the targets as in prior work [22, 30].

4.2 Defining a Recovery Set and Policy

Recovery RL executes a composite policy π in the environment, which selects between a task-driven policy π_{task} and a recovery policy π_{rec} at each timestep based on whether the agent is in danger of constraint violations in the near future. To quantify this risk, we use Q^{π}_{risk} to construct a recovery set that contains state-action tuples from which π may not be able to avoid constraint violations. Then if the agent finds itself in the recovery set, it executes a learned recovery policy instead of π_{task} to navigate back to regions of the MDP that are known to be sufficiently safe. Specifically, define two complimentary sets: the safe set $\mathcal{T}^{\pi}_{\text{safe}}$ and recovery set $\mathcal{T}^{\pi}_{\text{rec}}$:

$$\mathcal{T}^{\pi}_{\text{safe}} = \{(s, a) \in \mathcal{S} \times \mathcal{A} : Q^{\pi}_{\text{risk}}(s, a) \leq \epsilon_{\text{risk}}\} \qquad \mathcal{T}^{\pi}_{\text{rec}} = \mathcal{S} \times \mathcal{A} \setminus \mathcal{T}^{\pi}_{\text{safe}}$$

We consider state-action tuple (s, a) safe if in state s after taking action a, executing π has a discounted probability of constraint violation less than ϵ_{risk}.

If the task policy π_{task} proposes an action $a^{\pi_{\text{task}}}$ at state s such that $(s, a^{\pi_{\text{task}}}) \notin \mathcal{T}^{\pi}_{\text{safe}}$, then a recovery action sampled from π_{rec} is executed instead of $a^{\pi_{\text{task}}}$. Thus, the recovery policy in Recovery RL can be thought of as projecting π_{task} into a safe region of the policy space in which constraint violations are unlikely. The recovery policy π_{rec} is also an RL agent, but is trained to minimize $\hat{Q}^{\pi}_{\phi,\text{risk}}(s, a)$ to reduce the risk of constraint violations under π. Let $a_t^{\pi_{\text{task}}} \sim \pi_{\text{task}}(\cdot|s_t)$ and $a_t^{\pi_{\text{rec}}} \sim \pi_{\text{rec}}(\cdot|s_t)$. Then π selects actions as follows:

$$a_t = \begin{cases} a_t^{\pi_{\text{task}}} & (s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{safe}} \\ a_t^{\pi_{\text{rec}}} & (s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{4}$$

Recovery RL acts as a filtering mechanism that aims to block proposed actions that are likely to lead to unsafe states, equivalent to modifying the environment that π_{task} operates in with new dynamics:

$$P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a) = \begin{cases} P(s'|s,a) & (s,a) \in \mathcal{T}^{\pi}_{\text{safe}} \\ P(s'|s,a^{\pi_{\text{rec}}}) & (s,a) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{5}$$

We train $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on samples from π since π_{task} is not executed directly in the environment, but is rather filtered through π.

Algorithm 1 Recovery RL

```
Require: $\mathcal{D}_{\text{offline}}$, task horizon $H$, number of episodes $N$
 1: Pretrain $\pi_{\text{rec}}$ and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on $\mathcal{D}_{\text{offline}}$                # Section 4.3
 2: $\mathcal{D}_{\text{task}} \leftarrow \emptyset$, $\mathcal{D}_{\text{rec}} \leftarrow \mathcal{D}_{\text{offline}}$
 3: $s_0 \leftarrow$ env.reset()
 4: for $i \in \{1, \dots N\}$ do
 5:     for $t \in \{1, \dots H\}$ do
 6:         if $C(s_t) = 1$ or is_terminal($s_t$) then
 7:             $s_t \leftarrow$ env.reset()
 8:         $a_t^{\pi_{\text{task}}} \sim \pi_{\text{task}}(\cdot|s_t)$                      # Query task policy
 9:                                              # Check if task policy will be unsafe
10:         if $(s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{rec}}$ then
11:             $a_t \sim \pi_{\text{rec}}(\cdot|s_t)$                       # Select recovery policy
12:         else
13:             $a_t = a_t^{\pi_{\text{task}}}$                             # Select task policy
14:         Execute $a_t$, observe $s_{t+1}$, $r_t = R(s_t, a_t)$, $c_t = C(s_t)$
15:                                                          # Relabel transition
16:         $\mathcal{D}_{\text{task}} \leftarrow \mathcal{D}_{\text{task}} \cup \{(s_t, a_t^{\pi_{\text{task}}}, s_{t+1}, r_t)\}$
17:         $\mathcal{D}_{\text{rec}} \leftarrow \mathcal{D}_{\text{rec}} \cup \{(s_t, a_t, s_{t+1}, c_t)\}$
18:         Train $\pi_{\text{task}}$ on $\mathcal{D}_{\text{task}}$, $\pi_{\text{rec}}$ and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on $\mathcal{D}_{\text{rec}}$  # Eq. 3
```

It is easy to see that the proposed recovery mechanism will shield the agent from regions in which constraint violations are likely if $\hat{Q}^{\pi}_{\phi,\text{risk}}$ is correct and executing π_{rec} reduces its value. However, this poses a potential concern: while the agent may be safe, how do we ensure that π_{task} can make progress in the *new* MDP defined in equation 6? Suppose that π_{task} proposes an unsafe action $a_t^{\pi_{\text{task}}}$ under $\hat{Q}^{\pi}_{\phi,\text{risk}}$. Then, Recovery RL executes a recovery action $a_t^{\pi_{\text{rec}}}$ and observes transition $(s_t, a_t^{\pi_{\text{rec}}}, s_{t+1}, r_t)$ in the environment. However, if π_{task} is updated with this observed transition, it will not learn to associate its proposed action ($a_t^{\pi_{\text{task}}}$) in the new MDP with r_t and s_{t+1}. As a result, π_{task} may continue to propose the same unsafe actions without realizing it is observing the result of an action sampled from π_{rec}. To address this issue, for training π_{task}, we *relabel all actions with the action proposed by* π_{task}. Thus, instead of training π_{task} with executed transitions (s_t, a_t, s_{t+1}, r_t), π_{task} is trained with transitions $(s_t, a_t^{\pi_{\text{task}}}, s_{t+1}, r_t)$. This ties into the interpretation of defining a safe MDP with dynamics $P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a)$ for π_{task} to act in since all transitions for training π_{task} are relabeled as if π_{task} was executed directly.

4.3 Offline Pretraining

To convey information about constraints before interaction with the environment, we provide the agent with a set of transitions $\mathcal{D}_{\text{offline}}$ that contain constraint violations for pretraining. While this requires violating constraints in the environment, a human may be able to carefully demonstrate these unsafe transitions in a relatively controlled manner (e.g. gently tipping over a glass) so that the robot does not need to accidentally learn them online (e.g. knocking the glass off the table). We pretrain $\hat{Q}^{\pi}_{\phi,\text{risk}}$ by minimizing Equation 3 over offline batches sampled from $\mathcal{D}_{\text{offline}}$. We additionally pretrain π_{rec} using the data in $\mathcal{D}_{\text{offline}}$. Note that any RL algorithm can be used to represent π_{task} while any off-policy RL algorithm can be used to learn π_{rec}. For some environments in which exploration is challenging, we utilize a separate set of task demonstrations to initialize π_{task} to expedite learning.

Recovery RL first pretrains $\hat{Q}^{\pi}_{\phi,\text{risk}}$ and recovery policy π_{rec} on a set of transitions $\mathcal{D}_{\text{offline}}$ containing constraint violations. During online RL training, the agent actually executes π, which is an algorithmic selection between policy π_{task} and π_{rec}. This process is summarized in Algorithm 1 and Figure 1.

4.4 Practical Implementation

Recovery Policy: In principle, any off-policy RL algorithm can be used to learn π_{rec}. In this paper, we explore both model-free and model-based RL algorithms to learn π_{rec}. For model-free recovery, we perform gradient descent on the safety critic $\hat{Q}^{\pi}_{\phi,\text{risk}}(s, \pi_{\text{rec}}(s))$, as in the popular off-policy RL algorithm DDPG [31]. For model-based recovery, we perform model predictive control (MPC) over a learned dynamics model f_θ. For lower dimensional tasks, we utilize the PETS algorithm from Chua

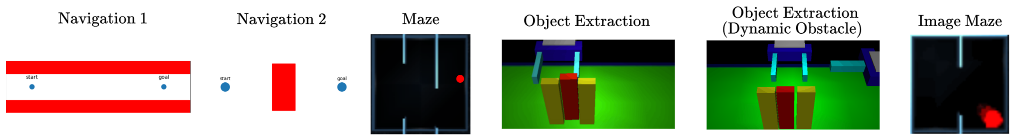

Figure 2: **Simulation Experiments Domains:** We evaluate Recovery RL on a set of 2D navigation tasks, two contact rich manipulation environments, and a visual navigation task. In Navigation 1 and 2, the goal is to navigate from the start set to the goal set without colliding into the obstacles (red) while in the Maze navigation tasks, the goal is to navigate from the left corridor to the red dot in the right corridor without colliding into walls/borders. In both object extraction environments, the objective is to grasp and lift the red block without toppling any of the blocks or colliding with the distractor arm (Dynamic Obstacle environment).

et al. [32] to plan over a learned stochastic dynamics model while for tasks with visual observations, we utilize a VAE based latent dynamics model. **Task Policy:** We utilize the popular maximum entropy RL algorithm SAC [30] to learn π_{task}, but note that any RL algorithm could be used. Details on the implementation of both policies can be found in the supplement.

5 Experiments

In the following experiments, we aim to study whether Recovery RL can (1) more effectively trade off task performance and constraint satisfaction than prior algorithms, which jointly optimize for both and (2) effectively leverage offline data for safe RL.

Domains: We evaluate Recovery RL on a set of 6 simulation domains (Figure 2) and an image-based constrained reaching task on a physical robot (Figure 4). All experiments involve policy learning under state space constraints, in which a constraint violation terminates the current episode. This makes learning especially challenging, since constraint violations directly preclude further exploration. This setting is reflective of a variety of real world environments, in which constraint violations can require halting the robot due to damage to itself or its surrounding environment.

We first consider three 2D navigation domains: Navigation 1, Navigation 2, and Maze. Here, the agent only observes its position in 2D space and experiences constraint violations if it hits obstacles, walls, or workspace boundaries. We then consider three higher dimensional tasks to evaluate whether Recovery RL can be applied to contact rich manipulation tasks (Object Extraction, Object Extraction (Dynamic Obstacle)) and vision-based continuous control (Image Maze). In the object extraction environments, the goals is to extract the red block without toppling any blocks, and in the case of Object Extraction (Dynamic Obstacle), also avoiding contact with a dynamic obstacle which moves in and out of the workspace. Image Maze is a shorter horizon version of Maze, but the agent is only provided with image observations rather than its (x, y) position in the environment.

We then evaluate Recovery RL on an image-based constrained reaching task on the da Vinci Research Kit (dVRK) [33] where the robot must guide its end effector within 2 mm of a target position from two possible starting locations while avoiding a stay-out zone for the end effector in the center of the workspace. The dVRK is cable-driven and has relatively imprecise controls, motivating closed-loop control strategies to compensate for these errors [34]. Furthermore, the dVRK system has been used in the past to evaluate safe RL algorithms [18] due to its high cost and the delicate structure of its arms, which make safe learning critical. Exact environment, task, and data collection details can be found in the supplement for all simulation and physical experiments.

Evaluation Metric: Since Recovery RL and prior methods trade off between safety and task progress, we report the ratio of the cumulative number of task successes and the cumulative number of constraint violations at each episode to illustrate this (higher is better). We tune all algorithms to maximize this ratio, and task success is determined by defining a goal set in the state space for each environment. To avoid issues with division by zero, we add 1 to the cumulative task successes and constraint violations when computing this ratio. This metric provides a single scalar value to quantify how efficiently different algorithms balance task completion and constraint satisfaction. We do not report reward per episode, as episodes terminate on task completion or constraint violation. In the supplementary material, we also report additional metrics for each experiment: cumulative task successes and cumulative constraint violations. For all experiments, we replicate each run across 3 random seeds and report the mean and standard error.

Comparisons: We compare Recovery RL to algorithms which ignore constraints (Unconstrained) and enforce constraints by implementing constraints into the policy optimization objective (LR, SQRL,

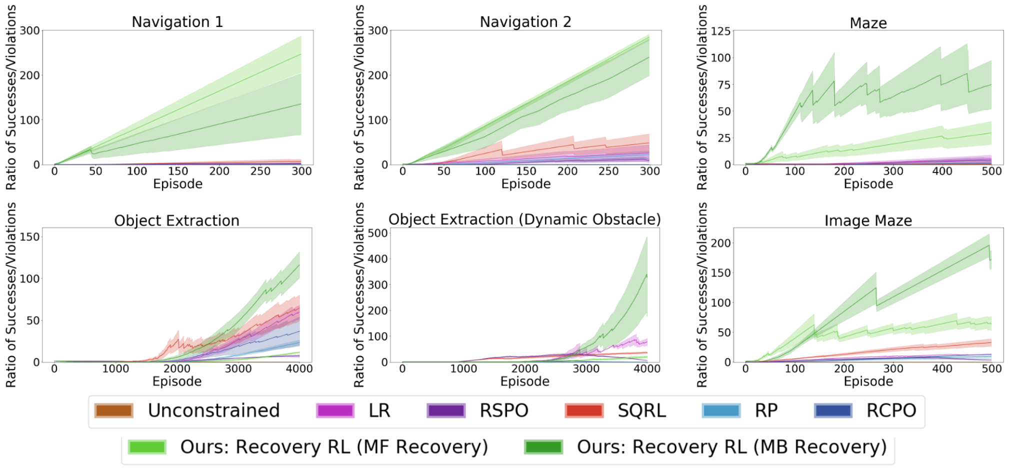

Figure 3: **Simulation Experiments:** In all navigation tasks, we find that Recovery RL significantly outperforms prior methods with both model-free and model-based recovery policies, while for the object extraction environments, Recovery RL with a model-based recovery policy significantly outperforms prior algorithms while Recovery RL with a model-free recovery policy does not perform as well. We hypothesize that this is due to the model-based recovery mechanism being better able to compensate for imperfections in $\hat{Q}^{\pi}_{\phi,\text{risk}}$. The sawtooth pattern occurs due to constraint violations, which result in a sudden drop in the ratio.

RSPO) or employing reward shaping (RP, RCPO). Specifically, we compare Recovery RL to: **Unconstrained**, where the agent only optimizes for the task reward and ignores constraints, **Lagrangian Relaxation (LR)**, which minimizes $L_{\text{policy}}(s,a,r,s';\pi)+\lambda(\mathbb{E}_{a\sim\pi(\cdot|s)}\left[\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a)\right]-\epsilon_{\text{risk}})$, where L_{policy} is the policy optimization loss function used and the second term approximately implements the constraint $\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a)\leq\epsilon_{\text{risk}}$, with both updated via dual gradient descent, **Safety Q-Functions for RL (SQRL) [22]**, which combines the LR method with a filtering mechanism to reject policy actions for which $\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a)>\epsilon_{\text{risk}}$, **Risk Sensitive Policy Optimization (RSPO)** [13], where the agent minimizes $L_{\text{policy}}(s,a,r,s';\pi)+\lambda_t(\mathbb{E}_{a\sim\pi(\cdot|s)}\left[\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a)\right]-\epsilon_{\text{risk}})$, where λ_t is a sequence that decreases to 0, **Reward Penalty (RP)**, in which the agent observes a new reward function that penalizes constraint violations: $R'(s,a)=R(s,a)-\lambda C(s)$, and **Critic Penalty Reward Constrained Policy Optimization (RCPO) [11]**, where the agent optimizes the Lagrangian relaxation via dual gradient descent and the policy gradient trick. The policy gradient update maximizes $\mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^t(R(s_t,a_t)-\lambda\hat{Q}^{\pi}_{\phi,\text{risk}}(s_t,a_t))\right]$ and the multiplier update is the same as in LR.

All of these algorithms are implemented with the same base algorithm for learning the task policy (Soft Actor Critic [30]) and all but Unconstrained and RP are modified to use the same safety critic $\hat{Q}^{\pi}_{\phi,\text{risk}}$ which is pretrained on $\mathcal{D}_{\text{offline}}$ for all methods. Thus, the key difference between Recovery RL and prior methods is how $\hat{Q}^{\pi}_{\phi,\text{risk}}$ is utilized: the comparisons use a joint objective which uses $\hat{Q}^{\pi}_{\phi,\text{risk}}$ to train a single policy that optimizes for both task performance and constraint satisfaction, while Recovery RL separates these objectives across two sub-policies. We tune all prior algorithms and report the best hyperparameter settings found on each task for the ratio based evaluation metric introduced above. See the supplement for ablations studying different hyperparameter choices for Recovery RL and the comparison algorithms, a detailed study of the importance of each component of Recovery RL, and further details on experimental setup and parameters.

5.1 Experiments

Simulation Experiments: We study the performance of Recovery RL and prior methods in all simulation domains in Figure 3. Results suggest that Recovery RL with both model-free and model-based recovery mechanisms significantly outperform prior algorithms across all 3 2D pointmass navigation environments (Navigation 1, Navigation 2, Maze) and the visual navigation environment (Image Maze). In the Object Extraction environments, we find that Recovery RL with model-based recovery significantly outperforms prior algorithms, while Recovery RL with a model-free recovery mechanism does not perform nearly as well. We hypothesize that the model-based recovery

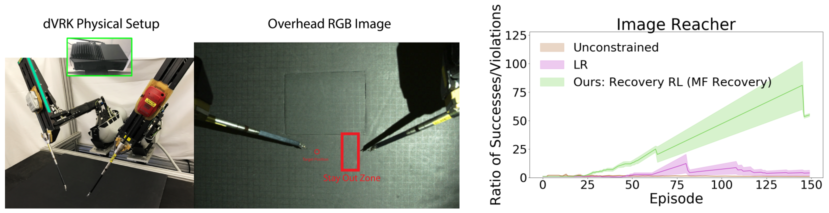

Figure 4: **Physical Experiment**: We evaluate Recovery RL on a constrained image-based reacher task on the dVRK with a stay out zone in the center of the workspace. We supply all algorithms with an overhead RGB image as input and find that Recovery RL significantly outperforms Unconstrained and LR.

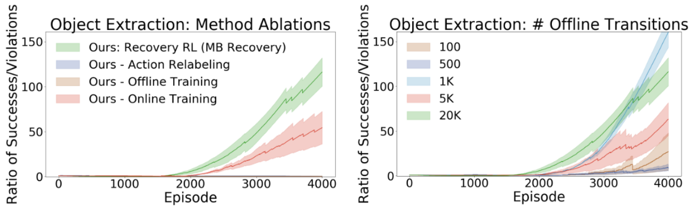

Figure 5: **Ablations:** We first study the affect of different algorithmic components of Recovery RL (left). Results suggest that offline pretraining of π_{rec} and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ is critical for good performance, while removing online updates leads to a much smaller reduction in performance. Furthermore, we find that the action relabeling method for training π_{task} (Section 4.2) is critical for good performance. We then study the sensitivity of Recovery RL with model-based recovery to the number of offline transitions used to pretrain π_{rec} and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ (right) and find that Recovery RL performs well even with just 1000 transitions in $\mathcal{D}_{\text{offline}}$ for the Object Extraction task, with performance degrading when the number of transitions is reduced beyond this point.

mechanism is better able to compensate for noise in $\hat{Q}^{\pi}_{\phi,\text{risk}}$, resulting in a more robust recovery policy. We find that the prior methods often struggle as they tend to sacrifice either safety or task performance, while Recovery RL is generally able to effectively optimize for task performance in the safe MDP defined by the recovery policy. We study this further in the supplement.

Physical Experiment: We evaluate Recovery RL and prior algorithms on an image-based reaching task with delta-position control on the da Vinci Research Kit in Figure 4. See Figure 4 for an illustration of the experimental setup. We find that Recovery RL substantially outperforms prior methods, suggesting that Recovery RL can be used for visuomotor control on physical robots.

Ablations: We ablate different components of Recovery RL and study the sensitivity of Recovery RL to the number of transitions in $\mathcal{D}_{\text{offline}}$ for the Object Extraction domain in Figure 5. Results suggest that Recovery RL performs much more poorly when π_{rec} and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ are not pretrained with data from $\mathcal{D}_{\text{offline}}$, indicating the value of learning to reason about safety before environment interaction. However, when π_{rec} and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ are not updated online, performance degrades much less significantly. A key component of Recovery RL is relabeling actions when training the task policy so that π_{task} can learn to associate its proposed actions with their outcome (Section 4.2). We find that without this relabeling, Recovery RL achieves very poor performance as it rarely achieves task successes. Additionally, we find that although the reported simulation experiments supply Recovery RL and all prior methods with $20,000$ transitions in $\mathcal{D}_{\text{offline}}$ for the Object Extraction task, Recovery RL is able to achieve good performance with just 1000 transitions in $\mathcal{D}_{\text{offline}}$, with performance significantly degrading only when the size of $\mathcal{D}_{\text{offline}}$ is reduced to less than this amount.

6 Conclusion

We present Recovery RL, a new algorithm for safe RL which is able to (1) efficiently leverage a small set of demonstrations of constraint violations to reduce the probability of constraint violations during learning and (2) effectively balance task-directed exploration and safety by decoupling them across a task policy and a recovery policy. We find that Recovery RL more effectively balances task performance and constraint satisfaction than 5 state-of-the-art prior algorithms for safe RL across 6 simulation domains and an image-based constrained reaching task on a physical robot. Results suggest that Recovery RL could scale well to robotic tasks with complex, contact rich dynamics and high dimensional state spaces such as images. In future work we hope to (1) explore further evaluation on physical robots, (2) establish formal guarantees, and (3) leverage ideas from offline RL to more effectively pretrain the recovery policy.

References

[1] C. Richter and N. Roy. Safe visual navigation via deep learning and novelty detection. *Robotics Science and Systems (RSS)*, 2013.

[2] 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. *CoRR*, 2018.

[3] 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. *Conference on Robot Learning (CoRL)*, 2018.

[4] A. Nagabandi, K. Konoglie, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. *Conference on Robot Learning (CoRL)*, 2019.

[5] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, , and C. J. Tomlin. A general safety framework for learning-based control in uncertain robotic systems. In *IEEE Transactions on Automatic Control*, 2018.

[6] 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.

[7] J. H. Gillula and C. J. Tomlin. Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor. 2012.

[8] S. Li and O. Bastani. Robust model predictive shielding for safe reinforcement learning with stochastic dynamics. 2020.

[9] A. Ray, J. Achiam, and D. Amodei. Benchmarking safe exploration in deep reinforcement learning. In *NeurIPS Deep Reinforcement Learning Workshop*, 2019.

[10] J. Achiam, D. Held, A. Tamar, and P. Abbeel. Constrained policy optimization. In *Proc. Int. Conf. on Machine Learning*, 2017.

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

[12] M. Heger. Consideration of risk in reinforcement learning. In *Machine Learning Proceedings*, 1994.

[13] Y. Shen, M. J. Tobia, T. Sommer, and K. Obermayer. Risk-sensitive reinforcement learning. In *Neural Computation*, volume 26, 2014.

[14] A. Tamar, Y. Glassner, and S. Mannor. Policy gradients beyond expectations: Conditional value-at-risk. In *CoRR*, 2014.

[15] Y. C. Tang, J. Zhang, and R. Salakhutdinov. Worst cases policy gradients. *Conf. on Robot Learning (CoRL)*, 2019.

[16] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. *International Conference on Learning Representations*, 2018.

[17] J. F. Fisac, N. F. Lugovoy, V. Rubies-Royo, S. Ghosh, and C. J. Tomlin. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2019.

[18] 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. *Robotics and Automation Letters (RAL)*, 2020.

[19] F. Berkenkamp, M. Turchetta, A. P. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Proc. Advances in Neural Information Processing Systems*, 2017.

[20] M. Turchetta, F. Berkenkamp, and A. Krause. Safe exploration in finite markov decision processes with gaussian processes. In *Proc. Advances in Neural Information Processing Systems*, 2016.

[21] F. W. Peterr Geibel. Risk-sensitive reinforcement learning applied to control under constraints. In *Journal of Artificial Intelligence Rersearch*, volume 24, 2005.

[22] K. Srinivasan, B. Eysenbach, S. Ha, J. Tan, and C. Finn. Learning to be safe: Deep reinforcement learning with a safety critic. *CoRR*, 2020. Available at https://drive.google.com/file/d/15x0YGKjBboz1DL3KZhm2xFnL-DtqslLT/view?usp=sharing.

[23] Y. Chow, O. Nachum, E. Duéñez-Guzmán, and M. Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. In *nips*, 2018.

[24] Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. Duéñez-Guzmán. Lyapunov-based safe policy optimization for continuous control. In *CoRR*, 2019.

[25] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu. Safe reinforcement learning via shielding. 2018.

[26] W. Han, S. Levine, and P. Abbeel. Learning compound multi-step controllers under unknown dynamics. *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015.

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

[28] 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. In *Workshop on the Algorithmic Foundations of Robotics*, 2020.

[29] E. Altman. *Constrained Markov Decision Processes*. 1999. doi:10.1016/0167-6377(96)00003-X.

[30] 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.

[31] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *Proc. Int. Conf. on Learning Representations*, 2016.

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

[33] 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.

[34] M. Hwang, B. Thananjeyan, S. Paradis, D. Seita, J. Ichnowski, D. Fer, T. Low, and K. Goldberg. Efficiently calibrating cable-driven surgical robots with rgbd sensing, temporal windowing, and linear and recurrent neural network compensation. *Robotics and Automation Letters (RAL)*, 2020.

[35] P. Tandon. Pytorch implementation of soft actor critic. https://github.com/pranz24/pytorch-soft-actor-critic, 2020.

[36] Q. Vuong. Pytorch implementation of pets. https://github.com/quanvuong/handful-of-trials-pytorch, 2020.

[37] S. Nair, S. Savarese, and C. Finn. Goal-aware prediction: Learning to model what matters. *Proc. Int. Conf. on Machine Learning*, 2020.

[38] 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.

Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones Supplementary Material

The supplementary material is structured as follows: In Section A we discuss brief theoretical motivation for Recovery RL and possible variants and in Section B we discuss algorithmic details for Recovery RL and comparison algorithms. In Section C, we report additional metrics for all domains and comparisons and in Section D, we report results for additional ablation studies evaluating hyperparameter sensitivity of Recovery RL. We qualitatively demonstrate the sensitivity by providing visualizations of the safety critic in Section E. We provide additional details about algorithm implementation in Section F, and on domain implementation in Section G. Finally, we report domain-specific algorithm hyperparameters in Section H.

A Recovery RL Theoretical Motivation and Variants

In this section, we will briefly and informally discuss additional properties of Recovery RL and then discuss some variants of Recovery RL.

A.1 Theoretical Motivation

Recall from Section 4, that the task policy is operating in an environment with modified dynamics:

$$P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a) = \begin{cases} P(s'|s,a) & (s,a) \in \mathcal{T}^{\pi}_{\text{safe}} \\ P(s'|s,a^{\pi_{\text{rec}}}) & (s,a) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{6}$$

However, $P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}$ changes over time (even within the same episode) and analysis of policy learning in non-stationary MDPs is currently challenging and ongoing work. Assuming that $P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}$ is stationary following the pretraining phase, it is immediate that π_{task} is operating in a stationary MDP $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}, R(\cdot,\cdot), \gamma, \mu)$, and therefore all properties of π_{task} in stationary MDPs apply in $\mathcal{M}'$. Observe that iterative improvement for π_{task} in $\mathcal{M}'$ implies iterative improvement for π in $\mathcal{M}$, since both MDPs share the same reward function, and an action taken by π_{task} in $\mathcal{M}'$ is equivalent to π_{task} trying the action in $\mathcal{M}$ before being potentially caught by π_{rec}.

A.2 Safety Value Function

One variant of Recovery RL can use a safety critic that is a state-value function $V^{\pi}_{\text{risk}}(s)$ instead of a state-action-value function. While this implementation is simpler, the Q^{π}_{risk} version used in the paper can switch to a safe action instead of an unsafe one instead of waiting to reach an unsafe state to start recovery behavior.

A.3 Reachability-based Variant

Another variant can use the learned dynamics model in the model-based recovery policy to perform a one (or k) step lookahead to see if future states-action tuples are in $\mathcal{T}^{\pi}_{\text{safe}}$. While Q^{π}_{risk} in principle carries information about future safety, this is an alternative method to check future states.

B Algorithm Details

B.1 Recovery RL

Recovery Policy: In principle, any off-policy reinforcement learning algorithm can be used to learn the recovery policy π_{rec}. In this paper, we explore both model-free and model-based reinforcement learning algorithms to learn π_{rec}. For model-free recovery, we perform gradient descent on the safety critic $\hat{Q}^{\pi}_{\phi,\text{risk}}(s, \pi_{\text{rec}}(s))$, as in the popular off-policy reinforcement learning algorithm DDPG [31]. We choose the DDPG-style objective function over alternatives since we do not wish the recovery policy to explore widely. For model-based recovery, we perform model predictive control (MPC) over a learned dynamics model f_θ by minimizing the following objective:

$$L_\theta(s_t, a_t) = \mathbb{E}\left[\sum_{i=0}^{H} \hat{Q}^{\pi}_{\phi,\text{risk}}(\hat{s}_{t+i}, a_{t+i})\right] \tag{7}$$

where $\hat{s}_{t+i+1} \sim f_\theta(\hat{s}_{t+i}, a_{t+i})$, $\hat{s}_t = s_t$, and $\hat{a} = a_t$. For lower dimensional tasks, we utilize the PETS algorithm from Chua et al. [32] to plan over a learned stochastic dynamics model while for tasks with visual observations, we utilize a VAE based latent dynamics model. In the offline pretraining phase, when model-free recovery is used, batches are sampled sequentially from $\mathcal{D}_{\text{offline}}$ and each batch is used to (1) train $\hat{Q}^\pi_{\phi,\text{risk}}$ by minimizing the loss in equation 3 and (2) optimize the DDPG policy to minimize the current $\hat{Q}^\pi_{\phi,\text{risk}}$. When model-based recovery is used, the data in $\mathcal{D}_{\text{offline}}$ is first used to learn dynamics model f_θ using either PETS (low dimensional tasks) or latent space dynamics (image-based tasks). Then, $\hat{Q}^\pi_{\phi,\text{risk}}$ is separately optimized to minimize the loss from equation 3 over batches sampled from $\mathcal{D}_{\text{offline}}$. During the online RL phase, all methods are updated online using on-policy data from composite policy π.

Task Policy: In experiments, we utilize the popular maximum entropy RL algorithm SAC [30] to learn π_{task}, but note that any RL algorithm could be used to train π_{task}. In general π_{task} is only updated in the online RL phase. However, in certain domains where exploration is challenging, we pre-train SAC on a small set of task-specific demonstrations to expedite learning. To do this, like for training the model-free recovery policy, we sample batches sequentially from $\mathcal{D}_{\text{offline}}$ and each batch is used to (1) train $\hat{Q}^\pi_{\phi,\text{risk}}$ by minimizing the loss in equation 3 and (2) optimize the SAC policy to minimize the current $\hat{Q}^\pi_{\phi,\text{risk}}$. To ensure that π_{task} learns which actions result in recovery behavior, we train π_{task} on transitions $(s_t, a_t^{\pi_{\text{task}}}, s_{t+1})$ even if π_{rec} was executed as noted in Section 4.2.

B.2 Unconstrained

We use an implementation of the popular model-free reinforcement learning algorithm Soft Actor Critic [35, 30], which maximizes a combination of task reward and policy entropy with a stochastic actor function.

B.3 Lagrangian Relaxation (LR)

In this section we will briefly motivate and derive the Lagrangian relaxation baseline. As before, we desire to solve the following constrained optimization problem:

$$\min_\pi L_{\text{policy}}(s;\pi) \text{ s.t. } \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi_{\text{risk}}(s,a)\right] \leq \epsilon_{\text{risk}}$$

where L_{policy} is a policy loss function we would like to minimize (e.g. from SAC). As in prior work in solving constrained optimization problems, we can solve the following unconstrained problem instead:

$$\max_{\lambda\geq 0}\min_\pi L_{\text{policy}}(s;\pi) + \lambda(\mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q^\pi_{\text{risk}}(s,a)\right] - \epsilon_{\text{risk}})$$

We aim to find a saddle point of the Lagrangian function via dual gradient descent. In practice, we use samples to approximate the expectation in the objective by sampling an action from $\pi(\cdot|s)$ each time the objective function is evaluated.

B.4 Risk Sensitive Policy Optimization (RSPO)

We implement Risk Sensitive Policy Optimization by implementing the Lagrangian Relaxation method as discussed in Section B.3 with a sequence of multipliers which decrease over time. This encourages initial constraint satisfaction followed by gradual increase in prioritization of the task objective and is inspired by the Risk Sensitive Q-learning algorithm from [13].

B.5 Safety Q-Functions for Reinforcement Learning (SQRL)

This baseline is identical to LR, except it additionally adds a Q-filter, that performs rejection sampling on the policy's distribution $\pi(\cdot|s_t)$ until it finds an action a_t such that $Q^\pi_{\text{risk}}(s_t, a_t) \leq \epsilon_{\text{risk}}$.

B.6 Reward Penalty (RP)

The reward penalty comparison simply involves subtracting a constant penalty λ from the task reward function when a constraint is violated. This is the only comparison algorithm other than Unconstrained which does not use the learned Q^π_{risk} or the constraint demos, but is included due to its surprising efficacy and simplicity.

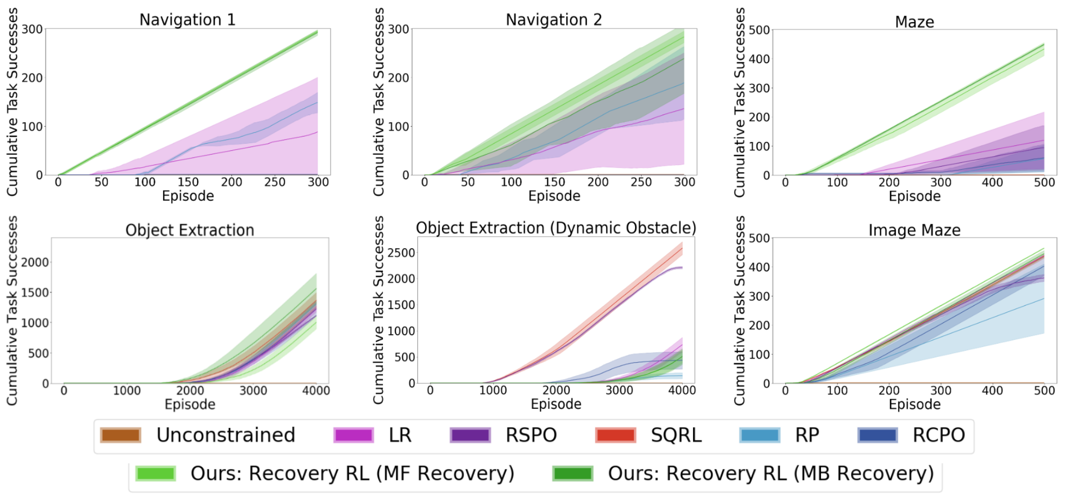

Figure 6: **Simulation Experiments Cumulative Successes:** We plot the cumulative task successes for each algorithm in each simulation domain and observe that Recovery RL (green), is generally among the most successful algorithms. In the cases that it has lower successes, we observe that it is safer (Figure 7). We find that Recovery RL has a higher or comparable task success rate to the next best algorithm on all environments except for the Object Extraction (Dynamic Obstacle) environment.

Table 1: **Constraint Violations Breakdown:** We report what percentage of constraint violations for each environment occur when the recovery policy is activated. If most constraint violations occur when the recovery policy is active, this indicates that the safety critic is likely relatively accurate while if this is not the case, it is likely that most constraint violations are due to imperfections in the safety critic itself rather than the recovery policy. We note that if the safety critic detects the need for recovery behavior too late, then these errors will be attributed to the recovery policy here. We find that for the low dimensional Maze and both Object Extraction environments, most constraint violations occur when the recovery policy is activated. In Navigation 1, none occur when the recovery policy is executed, but in this environment constraints are almost never violated by Recovery RL. In the Image Maze and Image Reacher tasks, we find that most violations occur when the recovery policy is not activated, which indicates that the bottleneck in these tasks is the quality of the safety critic. In Navigation 2, Recovery RL never violates constraints and only model-free recovery was run for Recovery RL on the Image Reacher task on the physical robot.

	Navigation 1	**Navigation 2**	**Maze**	**Object Extraction**	**Object Extraction (Dynamic Obstacle)**	**Image Maze**	**Image Reacher**
MF Recovery	0.00 ± 0.00	N/A	0.936 ± 0.059	0.992 ± 0.004	0.943 ± 0.012	0.194 ± 0.100	0.000 ± 0.000
MB Recovery	0.00 ± 0.00	N/A	0.938 ± 0.063	0.944 ± 0.055	0.833 ± 0.167	0.167 ± 0.167	N/A

B.7 Off Policy Reward Constrained Policy Optimization (RCPO)

In on-policy RCPO [11], the policy is optimized via policy gradient estimators by maximizing $\mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\left(\gamma^t R(s,a) - \lambda \gamma_{\text{risk}}^t D(s,a)\right)\right]$. In this work, we use $D(s,a) = Q_{\text{risk}}^{\pi}(s,a)$ and update the Lagrange multiplier λ as in LR. We could also use $D(s,a) = C(s)$, which would be almost identical to the RP baseline. Instead of optimizing this with on-policy RL, we use SAC to optimize it in an off-policy fashion to be consistent with the other comparisons.

C Additional Experimental Metrics

In Figure 6 and Figure 7, we report cumulative task successes and constraint violations for all methods for all simulation experiments. We report these statistics for the image reacher physical experiment in Figure 8. We observe that Recovery RL is generally very successful across most domains with relatively few violations. Some more successful comparisons tend to have many more constraint violations. We also report empirical probabilities for when constraint violations occur in Table 1, which suggests that in most tasks the recovery policy is already activated by Recovery RL when the violations do occur.

D Hyperparameter Sensitivity Experiments

We tune hyperparameters for Recovery RL and all prior methods to ensure a fair comparison. All algorithms are provided with the same offline data $\mathcal{D}_{\text{offline}}$. We first tune γ_{risk} and ϵ_{risk} for Recovery

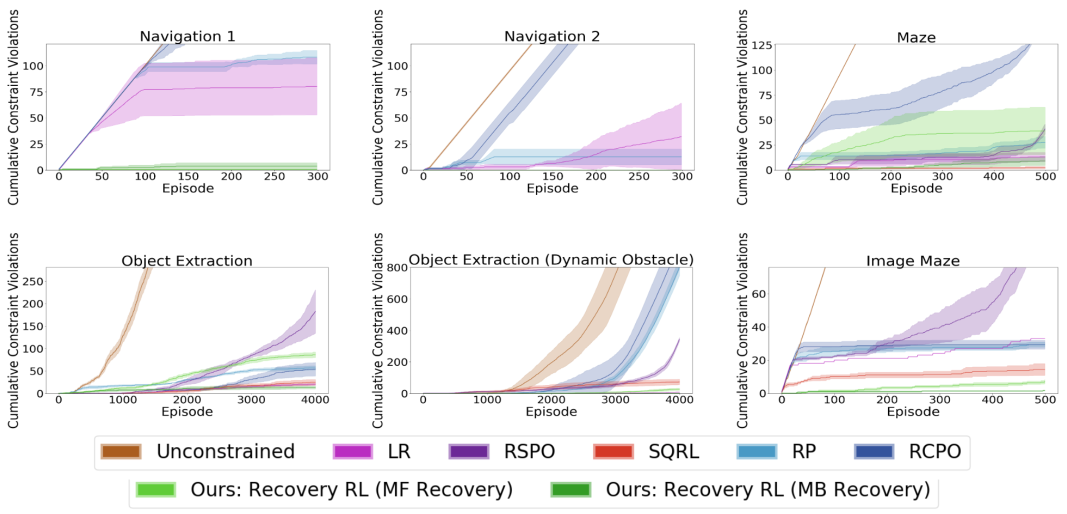

Figure 7: **Simulation Experiments Cumulative Violations:** We plot the cumulative constraint violations for each algorithm in each of the simulation domains and observe that Recovery RL (green), is among the safest algorithms across all domains. In the cases where it is less safe than a comparison, it has a higher task success rate (Figure 6).

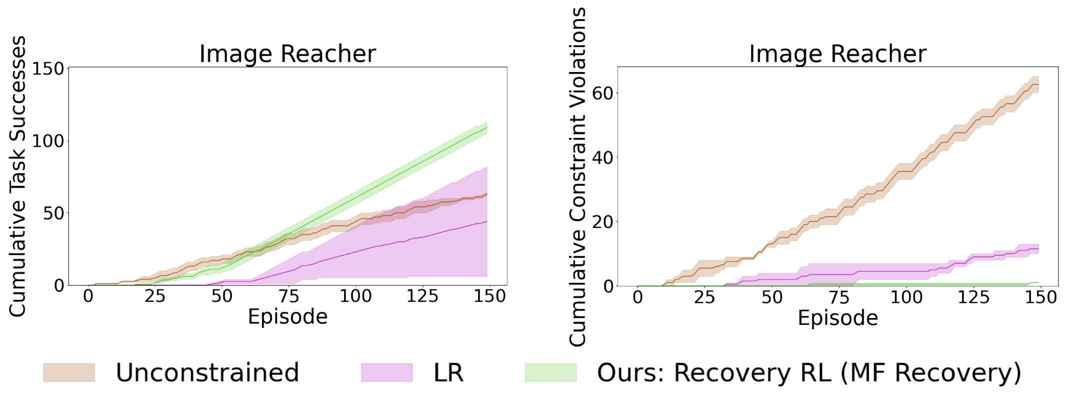

Figure 8: **Physical Experiments Successes and Violations:** We plot the cumulative constraint violations and task successes the image reacher task on the dVRK. We observe that Recovery RL is both more successful and safer than LR and unconstrained.

RL, and then use the same γ_{risk} and ϵ_{risk} for all other algorithms to ensure that all algorithms utilize the same training procedure for the safety critic. These two hyperparameters are the only two hyperparameters we tune for Recovery RL and SQRL. For the RP, RCPO, and LR comparisons we tune the penalty term λ with γ_{risk} and ϵ_{risk} fixed as mentioned above. For RSPO, we utilize a schedule which decays λ from 2 times the best value found for λ when tuning the LR baseline to 0 with an evenly spaced linear schedule over all training episodes.

In Figure 9, we study the sensitivity of Recovery RL with model-based recovery and the RP, RCPO, and LR comparisons to different hyperparameter choices on the Object Extraction task. For Recovery RL, we consider sensitivity to ϵ_{risk} over 3 values of γ_{risk} while for the comparison algorithms we consider sensitivity to the penalty term λ. We find that Recovery RL is less sensitive to hyperparameters than the other baselines for the γ_{risk} values we consider.

E Safety Critic Visualizations

We visualize the safety critic after pretraining for the navigation domains in Figure 10 and observe that increasing γ_{risk} results in a more gradual increase in regions near obstacles. Increasing γ_{risk} carries more information about possible future violations in $Q^{\pi}_{\text{risk}}(s, a)$. However, increasing γ_{risk} too much causes the safety critic to bleed too much throughout the state-action space as in the right-most column, making it difficult to distinguish between safe and unsafe states.

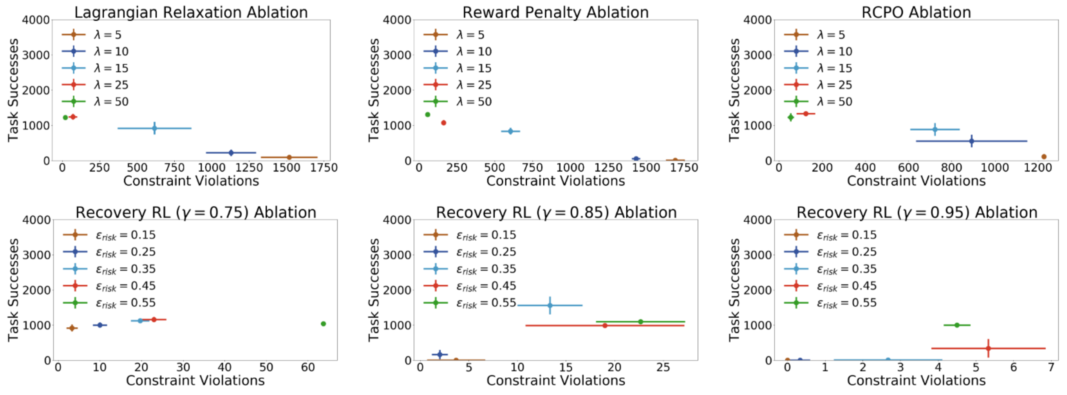

Figure 9: **Sensitivity Experiments:** We visualize the final number of task successes and constraint violations at the end of training for Recovery RL and comparison algorithms for a variety of different hyperparameter settings. We find that the comparison algorithms are relatively sensitive to the value of the penalty parameter λ while given a fixed γ_{risk}, Recovery RL achieves relatively few constraint violations while maintaining task performance over a range of ϵ_{risk} values.

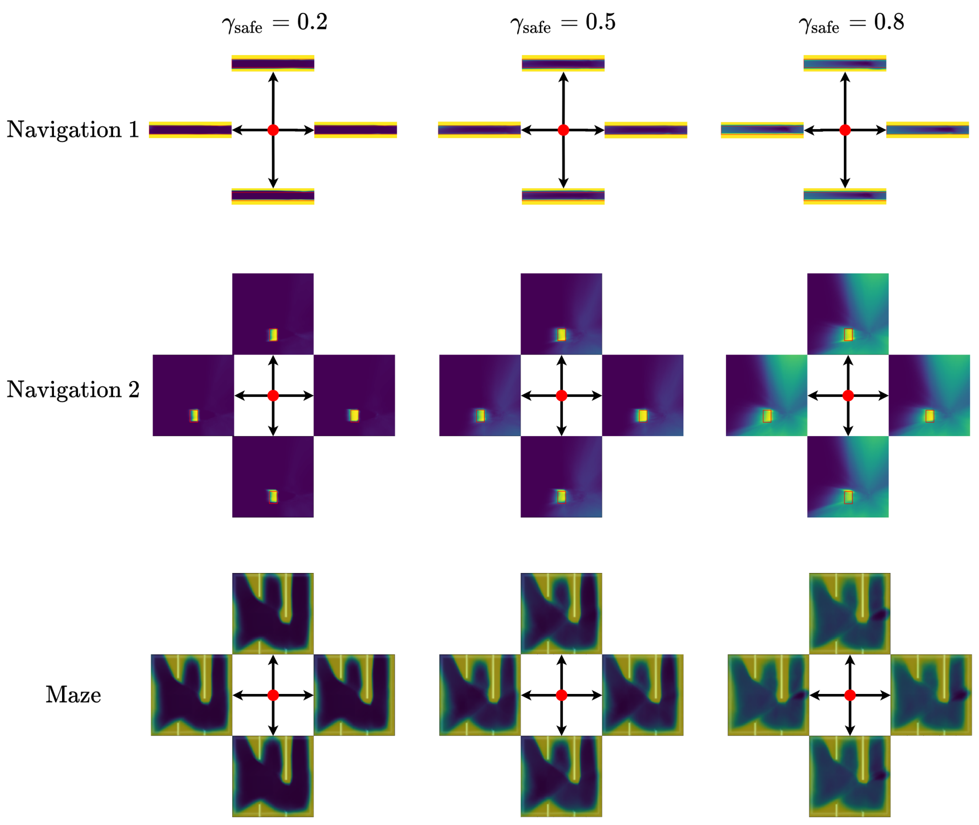

Figure 10: $\hat{Q}^{\pi}_{\phi,\text{risk}}$ **Visualization:** We plot the safety critic Q^{π}_{risk} for the navigation environments using the cardinal directions (left, right, up, down) as action input. We see that as γ_{risk} is increased, the gradient is lower, and the the function more gradually increases as it approaches the obstacles. Increasing γ_{risk} essentially increases the amount of information preserved from possible future constraint violations, allowing them to be detected earlier. These plots also illustrate action conditioning of the safety critic values. For example, the down action marks states as more unsafe than the up action directly above walls and obstacles.

F Implementation Details

Here we overview implementation and hyperparameter details for Recovery RL and all baselines. The recovery policy (π_{rec}) and task policy (π_{task}) are instantiated and trained in both the offline phase, in which data from $\mathcal{D}_{\text{offline}}$ is used to pre-train the recovery policy, and the online phase, in which Recovery RL updates the task policy with its exploration constrained by the learned safety critic and recovery policy. The safety critic and recovery policy are also updated online.

For all experiments, we build on the PyTorch implementation of Soft Actor Critic [2] provided in [35] and all trained networks are optimized with the Adam optimizer with a learning rate of $3e-4$. We first overview the hyperparameters and training details shared across Recovery RL and baselines in Section F.2 and then discuss the implementation of the recovery policy for Recovery RL in Section F.3.

F.1 Network Architectures

For low dimensional experiments, we represent the critic with a fully connected neural network with 2 hidden layers of size 256 each with ReLU activations. The policy is also represented with a fully connected network with 2 hidden layers of size 256 each, uses ReLU activations, and outputs the parameters of a conditional Gaussian. We use a deterministic version of the same policy for the model-free recovery policy. For image-based experiments, we represent the critic with a convolutional neural network with 3 convolutional layers to embed the input image and 2 fully connected layers to embed the input action. Then, these embeddings are concatenated and fed through two more fully connected layers. All fully connected layers have 256 hidden units each. We utilize 3 convolutional layers, with 128, 64, and 16 filters respectively. All layers utilize a kernel size of 3, stride of 2, and padding of 1. ReLU activations are used between all layers, and batch normalization units are added for the convolutional layers. For all algorithms which utilize a safety critic (Recovery RL, LR, SQRL, RSPO, RCPO), Q^{π}_{risk} is represented with the same architecture as the task critic except that a sigmoid activation is added at the output head to ensure that outputs are on $[0, 1]$ in order to effectively learn the probability of constraint violation. The task and model-free recovery policies also use the same architectures for image-based experiments, except that they output the parameters of a conditional Gaussian over the action space or an action, respectively.

F.2 Global Training Details

To prevent overestimation bias, we train two copies of all critic networks to compute a pessimistic (min for task critic, max for safety critic) estimate of the Q-values. Each critic is associated with a target network, and Polyak averaging is used to smoothly anneal the parameters of the target network. We use a replay buffer of size 1000000 and target smoothing coefficient $\tau = 0.005$ for all experiments except for the manipulation environments, in which a replay buffer of size 100000 and target smoothing coefficient $\tau = 0.0002$. All networks are trained with batches of 256 transitions. Finally, for SAC we utilize entropy regularization coefficient $\alpha = 0.2$ and do not update it online. We take a gradient step with batch size 1000 to update the safety critic after each timestep. We also update the model free recovery policy if applicable with the same batch at each timestep. If using a model-based recovery policy, we update it for 5 epochs at the end of each episode. For pretraining, we train the safety critic and model-free recovery policy for $10,000$ steps. We train the model-based recovery policy for 50 epochs.

F.3 Recovery Policy Training Details

In this section, we describe the neural network architectures and training procedures used by the recovery policies for all tasks.

F.3.1 Model-Free Recovery

The model-free recovery policy uses the same architecture as the task policy for all tasks, as described in Section F.1. However, it directly outputs an action in the action space instead of a distribution over the action space and greedily minimizes $\hat{Q}^{\pi}_{\phi,\text{risk}}$ rather than including an entropy regularization term as in [30]. The recovery policy is trained at each timestep on a batch of 1000 samples from the replay buffer.

F.3.2 Model-Based Recovery Training Details

For the non-image-based model-based recovery policy, we use PETS [32, 36], which trains and plans over a probabilistic ensemble of neural networks. We use an ensemble of 5 neural networks with 3 hidden layers of size 200 and swish activations (except at the output layer) to output the parameters of a conditional Gaussian distribution. We use the TS-∞ trajectory sampling scheme from Chua et al. [32] and optimize the MPC optimization problem with 400 samples, 40 elites, and 5 iterations for all environments. For image-based tasks, we utilize a VAE based latent dynamics model as in Nair et al. [37]. We train the encoder, decoder, and dynamics model jointly where the encoder and decoder and convolutional neural networks and the forward dynamics model is a fully connected network. We follow the same architecture as in Nair et al. [37]. For the encoder we utilize the following convolutional layers (channels, kernel size, stride): $[(32, 4, 2), (32, 3, 1), (64, 4, 2), (64, 3, 1), (128, 4, 2), (128, 3, 1), (256, 4, 2), (256, 3, 1)]$ followed by fully connected layers of size $[1024, 512, 2L]$ where L is the size of the latent space (predict mean and variance). All layers use ReLU activations except for the last layer. The decoder takes a sample from the latent space of dimension L and then feeds this through fully connected layers $[128, 128, 128]$ which is followed by de-convolutional layers (channels, kernel size, stride): $[(128, 5, 2), (64, 5, 2), (32, 6, 2), (3, 6, 2)]$. All layers again use ReLU activations except for the last layer, which uses a Sigmoid activation. For the forward dynamics model, we use a fully connected network with layers $[128, 128, 128, L]$ with ReLU activations on all but the final layer.

G Environment Details

In this section, we provide additional details about each of the environments used for evaluation.

G.1 Navigation Environments

1. **Navigation 1 and 2:** This environment has single integrator dynamics with additive Gaussian noise sampled from $\mathcal{N}(0, \sigma^2 I_2)$ where $\sigma = 0.05$ and drag coefficient 0.2. The start location is sampled from $\mathcal{N}\left((-50, 0)^\top, I_2\right)$ and the task is considered successfully completed if the agent gets within 1 unit of the origin. We use negative Euclidean distance from the goal as a reward function. Methods that use a safety critic are given 8000 transitions of data for pretraining.
2. **Maze:** This environment is implemented in MuJoCo and we again use negative Euclidean distance from the goal as a reward function. Methods that use a safety critic are given $10,000$ transitions of data for pretraining.

G.2 Manipulation Environments

We build two manipulation environments on top of the cartgripper environment in the visual foresight repository [38]. The robot can translate in cardinal directions and open/close its grippers.

1. **Object Extraction:** This environment is implemented in MuJoCo, and the reward function is -1 until the object is grasped and lifted, at which point it is 0 and the episode terminates. Constraint violations are determined by checking whether any object's orientation is rotated about the x or y axes by at least 15 degrees. All methods that use a safety critic are given $20,000$ transitions of data for pretraining. All methods are given 1000 transitions of task demonstration data to pretrain the task policy's critic function.
2. **Object Extraction (Dynamic Obstacle):** This environment is implemented in MuJoCo, and the reward function is -1 until the object is grasped and lifted, at which point it is 0 and the episode terminates. Constraint violations are determined by checking whether any object's orientation is rotated about the x or y axes by at least 15 degrees. Additionally, there is a distractor arm that is moving back and forth in the workspace in a periodic fashion. Arm collisions are also considered constraint violations. All methods that use a safety critic are given $20,000$ transitions of data for pretraining. All methods are given 1000 transitions of task demonstration data to pretrain the task policy's critic function.

G.3 Image Maze

This maze is also implemented in MuJoCo with different walls from the maze that has ground-truth state (Figure 2). Constraint violations occur if the robot collides with a wall. All methods are only supplied with RGB images as input, and all methods that use the safety critic are supplied with $20,000$ transitions for pretraining.

G.4 Physical Experiments

Physical experiments are run on the da Vinci Research Kit (dVRK) [33], a cable-driven bilateral surgical robot. Observations are recorded and supplied to the policies from a Zivid OnePlus RGBD camera. However, we only use RGB images, as the capture rate is much faster. End effector position is checked *by the environment* using the robot's odometry to check constraint violations and task completion, but this is not supplied to any of the policies. In practice, the robot's end effector position can be slightly inaccurate due to cabling effects such as hysteresis [34], but we ignore these effects in this paper. All methods that use a safety critic are supplied with $10,000$ transitions of data for pretraining, which takes around 3 hours to collect. To reduce extrapolation errors during learning, we sample a start state on the right side of the obstacle with probability 0.5 and sample one on the left side of the obstacle otherwise, as depicted in Figure 4.

H Environment Specific Algorithm Parameters

We use the same γ_{risk} and ϵ_{risk} for LR, RSPO, SQRL, and RCPO. For LR, RSPO, and SQRL, we find that the initial choice of λ strongly affects the overall performance of this algorithm and heavily tune this. We use the same values of λ for LR and SQRL, and use twice the best value found for LR in as an initialization for the λ-schedule in RSPO. We also heavily tune λ for RP and RCPO. These values are shown for each environment in the tables below.

Algorithm Name	Hyperparameter Format
LR	$(\gamma_{\text{risk}}, \epsilon_{\text{risk}}, \lambda)$
RP	λ
RCPO	$(\gamma_{\text{risk}}, \epsilon_{\text{risk}}, \lambda)$
MF Recovery	$(\gamma_{\text{risk}}, \epsilon_{\text{risk}})$
MB Recovery	$(\gamma_{\text{risk}}, \epsilon_{\text{risk}}, H)$

	LR	RP	RCPO	MF Recovery	MB Recovery
Navigation 1	$(0.8, 0.3, 5000)$	1000	$(0.8, 0.3, 1000)$	$(0.8, 0, 3)$	$(0.8, 0.3, 5)$
Navigation 2	$(0.65, 0.1, 1000)$	3000	$(0.65, 0.1, 5000)$	$(0.65, 0, 1)$	$(0.65, 0.1, 5)$
Maze	$(0.5, 0.15, 100)$	50	$(0.5, 0.15, 50)$	$(0.5, 0, 15)$	$(0.5, 0.15, 15)$
Object Extraction	$(0.75, 0.25, 50)$	50	$(0.75, 0.25, 50)$	$(0.75, 0, 25)$	$(0.85, 0.35, 15)$
Object Extraction (Dyn. Obstacle)	$(0.85, 0.25, 20)$	25	$(0.85, 0.25, 10)$	$(0.85, 0.35)$	$(0.85, 0.25, 15)$
Image Maze	$(0.65, 0.1, 10)$	20	$(0.65, 0.1, 20)$	$(0.65, 0, 1)$	$(0.6, 0.05, 10)$
Image Reacher	$(0.55, 0.05, 1000)$	N/A	N/A	$(0.55, 0.05)$	N/A