Title
Safe Reinforcement Learning Using Learned Safe Sets

Author
Thananjeyan, Brijen

Publication Date
2022

Safe Reinforcement Learning Using Learned Safe Sets

by

Brijen Abhimanyu Thananjeyan

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Ken Goldberg, Co-chair
Professor Joseph E. Gonzalez, Co-chair
Professor Francesco Borrelli

Spring 2022

Safe Reinforcement Learning Using Learned Safe Sets

Abstract

Safe Reinforcement Learning Using Learned Safe Sets

by

Brijen Abhimanyu Thananjeyan

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Ken Goldberg, Co-chair

Professor Joseph E. Gonzalez, Co-chair

Reinforcement learning is an increasingly popular framework that enables robots to learn to perform tasks from prior experience in environments where dynamics or shaped reward functions are challenging to model. However, because this requires robots to sample trajectories under significant dynamical uncertainty, the robot may perform unsafe maneuvers during online exploration. This is particularly problematic in real-world robotics, where unsafe behaviors can lead to damage to surroundings. As a result, many impressive reinforcement learning results are in simulation only. Safe reinforcement learning is a field with a rich history that studies how to reduce the number and magnitude of unsafe behaviors during learning, particularly in the real world. Safe reinforcement learning is challenging, because it requires limiting exploration to provide safety, but enabling sufficient exploration to maximize the task reward function. Algorithms frequently draw inspiration from methods in control theory, constrained optimization, and online learning to adaptively balance task-driven exploration and safety based on prior experience.

This thesis presents a set of novel safe reinforcement learning algorithms that maintain subsets of the state space where safety is highly probable under the current policy. The algorithms leverage these safe sets in different ways to promote safety during online exploration in the real world. The first part of the thesis covers a class of algorithms that requires the robot to maintain a conservative safe set of states from which it has already completed the task. As long as the robot approximately maintains the ability to return to the safe set, the robot can explore outside the safe set and iteratively expand it. This thesis also presents strong theoretical guarantees for this class of algorithms under known but stochastic, nonlinear dynamics. The second part presents another class of algorithms that maintains a much larger safe set based on the probability of the robot committing unsafe behaviors. The robot uses the boundary of this set to determine whether it should focus on task-driven exploration or safety recovery maneuvers. The final part of this thesis covers an algorithm that uses policy uncertainty to implicitly model safety and request human interventions for corrective feedback. This thesis concludes with a commentary on lessons learned and future endeavors.

To all of my teachers.

Contents

Acknowledgments

Completing a PhD in computer science at UC Berkeley is not an outcome I would have predicted a decade ago. This would not have been possible without the influence of many individuals and circumstances in my life. Joining a FIRST robotics team as a freshman in high school, I did not know that these experiences would propel me into a career in robotics research. I would like to thank all of the members of FTC Team 5151 Infinity Robotics for the memorable experiences working into the morning on an extremely imperfect robot that we built and programmed with no external assistance. While we did not have had the resources that other teams had access to, this makes me prouder of what we accomplished over the years. This adventure is why I continue to work with robots today, as difficult as they can be sometimes, and I owe you for that.

I would like to thank UC Berkeley and the UC Berkeley EECS department for teaching me much of what I know today. The undergraduate curriculum in Berkeley EECS transformed my way of thinking well beyond academia and engineering, and I am extremely grateful that I had the privilege of this educational opportunity. The tireless enthusiasm and support of professors, GSIs, and classmates made my UC Berkeley education a collaborative and growth-fostering experience that inspired me to continue towards a PhD in computer science. I will carry the lessons learned from my experience at Berkeley for the rest of my life.

I would like to thank my advisor Professor Ken Goldberg for taking me into his lab in early 2016 and exposing me to the vast world of robotics research. Thank you for all of the feedback on research proposals, paper drafts, and presentations you have provided over the last six and a half years. I truly appreciate that you always gave thoughtful, thorough, and honest feedback, and this significantly increased the quality of everything I did. I appreciate all of the time you have made for me over the years. Thanks to you, I am leaving Berkeley better equipped to formulate, tackle, and present my future ideas.

I would also like to thank my advisor Professor Joey Gonzalez. Thank you for taking me into your group in 2018, and I really appreciate all of the brainstorming sessions we have had over the years. Countless ideas from our discussions have helped shape the many projects and papers I have worked on. I also appreciate your listening and support when I was stuck or confused.

The most rewarding part of my PhD was the many collaborations I have been fortunate to have been a part of. I would like to first thank Ashwin Balakrishna, with whom I have co-authored twelve conference and journal papers and co-organized a successful NeurIPS workshop on safe reinforcement learning. It helped me greatly to have you to explore the challenges of PhD with. I remember bouncing terrible ideas off you every single day, until eventually they started to make sense. I admire your ability to stay calm and collected under pressure, your capacity to effectively manage countless projects and responsibilities, your eloquent presentation skills, and your ability to generate ideas nonstop. I hope that I will one day be as adept at these as you are. Thank you for the very rewarding collaborations, friendship, and for all of the future adventures we are bound to go on.

I am thankful for my close collaborations over several years with Jeff Ichnowski, Justin Kerr, Priya Sundaresan, Jenn Grannen, Minho Hwang, Daniel Seita, and Sam Paradis. I have learned many lessons from watching each of you closely. You each constantly inspire me to improve, and I hope we will have the opportunity to work together again someday.

I thank my many remaining co-authors and collaborators, including Adi Ganapathi, Aaron Ames, Animesh Garg, Ajay Tanwani, Francesco Borrelli, Izzy Putterman, Arsh Zahed, Albert Wilcox, Sophea Bonne, Sanjay Krishnan, Ugo Rosolia, Carolyn Matl, Chelsea Finn, Daniel S. Brown, Daniel Seita, Ion Stoica, Ellen Novoseller, Danyal Fer, Michael Luo, Lauren Miller, Jessica Ji, Kishore Srinivas, Johnathan Chiu, Krishnan Srinivasan, Jie Tan, Michael I. Jordan, Kirthevasan Kandasamy, Marek Petrik, Melanie Zeilinger, John Canny, Jonathan Lee, Julian Ibarz, Karthik Dharmarajan, Kaushik Shivakumar, Kevin Stone, Sergey Levine, Rowan McAllister, Ryan Hoque, Will Panitch, Jerri-Lynn Kincade, Thomas Kollar, Michael Laskey, Suraj Nair, Sylvia Herbert, Richard Liaw, Alejandro Escontrela, Thomas Low, Raven Huang, Roy Fox, Vatsal Patel, and Vainavi Viswanath. Thank you for all of the insightful discussions, shared deadline grinds, and fun times together. Lastly, I would like to thank all of my friends in the AUTOLab, BAIR, and the RISELab. I look forward to coming to campus because of you, and I will miss you greatly when it is officially time to leave.

Chapter 1

Introduction

Learning-based control and decision-making are active areas of research in robotics, and there is a heavy focus on methods that enable robots to learn online from their interactions in the world. Online learning enables robots to explore and try out new strategies, which can enable them to iteratively improve over time and adapt to changing scenarios. However, because the real-world dynamics of robotic manipulation are often very challenging to model, exploration strategies can operate under significant dynamical uncertainty which can lead to unsafe and catastrophic behaviors during learning. Due to this, most impressive reinforcement learning results are limited to simulation, to avoid the real world cost associated with unsafe actions. The rapidly-growing field of safe reinforcement learning focuses on providing theoretical abstractions and exploration strategies that address this challenge. Progress in safe reinforcement learning is a prerequisite to the deployment of online learning-based robots in safety-critical, *real-world* domains such as driving, hospitals, or households.

Before beginning, it is important to define the notion of safety used in this work. In this thesis, an *unsafe state* is a state that indicates an undesirable behavior, according to the user. Examples of unsafe states in practice could include states with broken objects, damage to surroundings, or entering a stay-out zone of the workspace. We will generally consider unsafe behaviors to be catastrophic offenses that terminate the current episode and require human intervention to reset the environment or correct. We will also interchangeably use the phrase *constraint violation* to refer to unsafe states. In general, states that are not unsafe are considered *safe.* The goal of a safe reinforcement learning algorithm is to learn to perform a desired behavior or task with high performance without encountering too many unsafe states during the learning process. We will mathematically formalize this idea in subsequent chapters.

Safe reinforcement learning is an area of research with a rich history, with methods dating back as early as the 1970s [1]. Many works, including this one, draw inspiration from prior work in constrained optimization, online learning, and control theory, and safety is considered in different, sometimes equivalent, ways.

This thesis proposes a class of safe reinforcement learning algorithms that attempt to achieve this by maintaining regions of safety in the workspace known as *safe sets.* In this work, different styles of safe sets will be learned from data and used during exploration to avoid unsafe behaviors. Each method will define a different type of safe set, which will be used differently during online exploration to either maintain safety guarantees, switch to recovery behaviors, or query human interventions. The common theme across these methods will be the existence of a contingency ”backup” plan, which can be executed if the robot is in a risky state. This work presents a combination of theoretical results and empirical experiments, including on physical robots, to analyze both the safety and sample efficiency of methods with respect to alternative approaches.

1.1 Background

This thesis covers algorithms that safely explore online in the state space by leveraging learned safe sets and focuses on algorithms that are designed for exploration in the real world. The algorithms presented in this thesis share many parallels and inspirations from works in safe reinforcement learning, control theory, and interactive imitation learning.

1.1.1 Safety in Reinforcement Learning

Safe reinforcement learning dates back as early as the 1970s [1], and a main focus is designing methods that avoid constraint violations during learning and online exploration in the real world [2]. Methods attempt to achieve this using a variety of means. Many methods impose safety during exploration via a Lagrangian function optimization that combines task-driven optimization with safety constraints in a single policy [3–7]. Many methods, including some of the ones previously mentioned also enforce safety during policy execution by learning a measure of safety and adapting the policy outputs based on it [4, 7–10]. The methods proposed in this work fall in the latter category and use learned regions of the state space with safety implications to adaptively select actions during online exploration.

1.1.2 Safe Sets and Shielding in Control Theory

Safe sets have been extensively studied in control theory and they can be defined in different ways. Prior work uses Hamilton-Jacobi reachability analysis to construct task and safety policies and a decision rule to select which one to execute [11, 12]. Other works define safe sets that are based on the abilities of prior safe policies and enforce that new policies do not explore too far away from their regions of confidence [13–16]. The algorithms described in this thesis are very similar in flavor, but do not assume access to known system dynamics. In contrast, the presented algorithms will use data to either perform online system identification and explicitly reason about dynamical uncertainty using stochastic modeling, or they will implicitly learn dynamical distance to dangerous behaviors via value function learning.

1.1.3 Human Interventions in Robot Learning

A common method to fall back to safety during online execution is to query a human supervisor. Human-gated imitation learning algorithms are an area of active research, where a human either decides to intervene or is called to intervene and take control of a robot [17–22]. These algorithms often maintain functions that estimate uncertainty [20, 22], supervisor discrepancy [17, 19, 22], or probability of failure, which are used to request a human intervention. The minimization of supervisor burden, or the work done by a human is also a desirable property of imitation learning algorithms [17, 19, 23]. Supervisor burden is often modeled by the length of interventions and also the number of context switches experienced by supervisors. This thesis presents a novel algorithm that solicits human interventions when predicted supervisor discrepancy is large and does not switch back until the robot is sufficiently confident, to jointly minimize both the context switching overhead of supervisors and the total length of interventions [17]

1.2 Thesis Structure

This thesis consists of three main parts. In Part I, I will introduce a subclass of safe reinforcement learning algorithms that maintains a very conservative safe set to which the robot must always maintain the ability to reach. The safe set considered in this subclass uses information about prior safe trajectories to structure exploration. The robot is allowed to explore outside of the safe set as long as it maintains this property, which enables it to guarantee safety under additional theoretical assumptions. This is in turn used to grow the safe set over time, giving the agent the ability to iteratively improve its performance as it becomes more comfortable in more regions of the state space. I will present a control-theoretic version of this algorithm in Chapter 2, which a suite of desirable theoretical guarantees for stochastic, nonlinear dynamical systems with *known* dynamics and input noise distributions. In Chapter 3, I will discuss how to relax this algorithm to apply in the reinforcement learning setting where system dynamics are unknown. This will remove the theoretical guarantees of the method, but empirical experiments suggest that the relaxation still provides some of these properties empirically on a set of benchmark simulation and physical experiments. This version of the algorithm is particularly designed for fully-observable, low-dimensional dynamical systems, so in Chapter 4, I will present an extension that scales to high-dimensional image observations using latent space safe sets. One limitation of the methods in this subsection are that they rely on computationally-expensive model-based planners during online execution. In Chapter 5, I will shift gears slightly and present an algorithm with theoretical guarantees that distills an offline, model-based reinforcement learning algorithm into an online, model-free policy.

In Part II, I will construct safe sets that, in contrast to the previous part, use prior trajectories containing unsafe behaviors to guarantee safety. While the work in this section will specifically focus on the reinforcement learning setting, it is heavily related and inspired by recent advances in the model predictive control shielding community. Chapter 6 presents an algorithm that leverages a offline demonstrations of unsafe behaviors to extract information about the location of constraint violating zones in the state space and the dynamics of the system near them. This information is used to learn a recovery zone, where the agent switches it focus from task-driven exploration to recovering to safety. The safe set in this section is the complement of the recovery zone and constraint violating regions, and it is much larger than the one considered in Part I. In Chapter 7, I will present an extension of this work that transfers safe sets from other, similar dynamical systems to a test environment using meta-learning.

In Part III, a different form of safe set will be introduced. In this section, the safe set will be based on policy uncertainty and not explicit safety modeling. Upon exiting the safe set, the robot will query a human supervisor, that will take control and provide corrective feedback, which will be used for policy updates.

Finally, I will conclude in Chapter 9 with remarks on lessons learned and future work.

.

Part I

Safe Exploration Using Prior Successes

Chapter 2

Constructing a Safe Set for Stochastic, Nonlinear Control Using Prior Successes

In this chapter, we will construct a learning-based policy for stochastic, nonlinear dynamics that is guaranteed to converge to a goal set, maintain safety during learning, and improve each iteration. In this chapter, we will assume the dynamics are known, but we will relax this assumption in subsequent chapters. The policy in this section maintains regions known as safe sets that denote regions of the state space where a prior policy can safely converge to the goal set. As long as the current policy iterate follows receding horizon plans that can robustly reach this set, it can also guarantee safe convergence to the goal set.

2.1 Introduction

Model Predictive Control (MPC) is an established control methodology which systematically uses forecasts to compute control actions. In MPC at each time t, the policy predicts a trajectory over a short time horizon, then it applies to the system to the first control action and the process is repeated at the next time step. This technique has seen significant success in a variety of robotic tasks [24–26], and there is substantial experimental and theoretical work demonstrating that the resulting closed loop system performs well on challenging tasks in stochastic dynamical systems [13, 16, 24, 27]. In this chapter, we build on the recently proposed learning model predictive control (LMPC) framework [13, 15, 16]. We assume a known stochastic dynamics model and design an iteratively improving MPC-based control strategy by estimating safe sets and value functions from past closed-loop trajectories.

The LMPC framework [13, 15, 16] presents a novel class of reference-free control strategies which utilize MPC to iteratively improve upon a suboptimal policy for a goal directed task. LMPC algorithms typically operate in the iterative learning control setting with fixed initial and terminal conditions, and provide robust guarantees on iterative improvement (in terms of task cost) for stochastic linear systems [13, 16] and deterministic nonlinear systems [15], if the MPC problem can be solved exactly. However, while LMPC-based control strategies exhibit a variety of desirable theoretical properties [13, 15, 16] and have been shown to work well on practical problems on physical robotic systems [24, 28], they have two key limitations: (1) guarantees for stochastic systems are limited to linear systems while practical systems are often stochastic and nonlinear and (2) start states and goal sets are typically assumed to be identical in each iteration.

We address both of these challenges. First, we extend the results in [16] to show iterative improvement guarantees for stochastic nonlinear systems. Second, we present a method to expand the set of feasible start states and goal sets during learning while maintaining these guarantees. Finally, we introduce sample-based approximations to present a practical algorithm to learn safe policies, which reliably complete tasks with varying boundary conditions while satisfying pre-specified constraints. In this chapter, we provided a more detailed treatment and theoretical analysis of the policy presented in Thananjeyan et al. [14] in addition to a novel method to expand the domain of the policy. This chapter presents (1) a novel multi-start, multi-goal LMPC algorithm, Adjustable Boundary Condition LMPC (ABC-LMPC), which optimizes expected costs subject to robust constraints, with (2) guarantees on expected performance, robust constraint satisfaction, and convergence to the goal for stochastic nonlinear systems, (3) a practical algorithm for expanding the allowed set of initial states and goal sets during learning, and (4) simulated continuous control experiments demonstrating that the learned policy can adapt to novel start states and goal sets while consistently and efficiently completing tasks during learning.

2.2 Related Work

Model Predictive Control: There has been a variety of prior work on learning based strategies for model predictive control in the reinforcement learning [24–27] and controls communities [29–35]. Learning strategies are used for estimating at least one of the following components needed to design MPC policies: *i*) a model of the system and the associated prediction accuracy [24–26, 28, 30, 32, 33], *ii*) a safe set of states from which the control task can be completed using a known safe policy [36–39] and *iii*) a value function [13, 16, 24, 40], which for a given safe policy, maps each state of the safe set to the closed-loop cost to complete the task. For an extensive survey of learning strategies used in MPC please refer to [41].

The most closely related works, both by Rosolia et. al. [13, 16], introduce the learning MPC framework for iterative learning control in stochastic linear systems. Here, MPC is used to iteratively improve upon a suboptimal demonstration by estimating a safe set and a value function from past closed loop trajectories. Robust guarantees are provided for iterative policy improvement if the MPC problem can be solved exactly. Furthermore, Thananjeyan et al. [24] propose a practical reinforcement learning algorithm using these strategies to learn policies for nonlinear systems. However, [13, 24] are limited to the iterative learning control setting, and although [16] presents a strategy for policy domain expansion, the method is limited to linear systems and requires the user to precisely specify an expansion direction. In this chapter, we build on this framework by (1) extending the theoretical results to prove that under similar assumptions, LMPC based policies yield iterative improvement in expectation under certain restrictions on the task cost function and (2) providing a practical and general algorithm to adapt to novel start states and goal sets while preserving all theoretical guarantees on policy performance.

Reinforcement Learning: There has been a variety of work from the reinforcement learning (RL) community on learning policies which generalize across a variety of initial and terminal conditions. Curriculum learning [42–44] has achieved practical success in RL by initially training agents on easier tasks and reusing this experience to accelerate learning of more difficult tasks. Florensa et al. [42] and Resnick et al. [43] train policies initialized near a desired goal state, and then iteratively increase the distance to the goal state as learning progresses. While these approaches have achieved practical success on a variety of simulated robotic and navigation tasks, the method used to expand the start state distribution is heuristic-based and requires significant hand-tuning. We build on these ideas by designing an algorithm which expands the start state distribution for an MPC-based policy by reasoning about reachability, similar to Ivanovic et al. [45]. However, [45] provides a curriculum for model free RL algorithms and does not provide feasibility or convergence guarantees, while we present an MPC algorithm which expands the set of allowed start states while preserving policy feasibility and convergence guarantees. There is also recent interest in goal-conditioned RL [46, 47]. The most relevant prior work in this area is hindsight experience replay [48], which trains a goal-conditioned policy using imagined goals from past failures. This strategy efficiently reuses data to transfer to new goal sets in the absence of dense rewards. We use a similar idea to learn goal-conditioned safe sets to adapt to novel goal sets by reusing data from past trajectories corresponding to goal sets reached in prior iterations.

Motion Planning: The domain expansion strategy of the proposed algorithm, ABC-LMPC, bears a clear connection to motion planning in stochastic dynamical systems [49, 50]. Exploring ways to use ABC-LMPC to design motion planning algorithms which can efficiently leverage demonstrations is an exciting avenue for future work, since the receding horizon planning strategy could prevent the exponential scaling in complexity with time horizon characteristic of open-loop algorithms [51]. We are also excited about exploring ways the domain expansion properties of ABC-LMPC can be used to facilitate safe navigation [52] and planning [53] in complex robotic systems.

2.3 Problem Statement

In this chapter, we consider nonlinear, stochastic, time-invariant systems of the form:

$$x_{t+1} = f(x_t, u_t, w_t) \tag{2.1}$$

where $x_t \in \mathbb{R}^n$ is the state at time t, $u_t \in \mathbb{R}^m$ is the control, $w_t \in \mathbb{R}^k$ is a disturbance input, and x_{t+1} is the next state. The disturbance w_t is sampled i.i.d. from a known distribution over a bounded set $\mathcal{W} \subseteq \mathbb{R}^p$. We denote Cartesian products with exponentiation, e.g. $\mathcal{W}^2 = \mathcal{W} \times \mathcal{W}$. We consider constraints requiring states to belong to the feasible state space $\mathcal{X} \subseteq \mathbb{R}^n$ and controls to belong to $\mathcal{U} \subseteq \mathbb{R}^m$. Let x_t^j, u_t^j, and w_t^j be the state, control input, and disturbance realization sampled at time t of iteration j respectively. Let $\pi^j : \mathbb{R}^n \to \mathbb{R}^m$ be the control policy at iteration j that maps states to controls (i.e. $u_t^j = \pi^j(x_t^j)$).

Unlike [16], in which the goal of the control design is to solve a robust optimal control problem, we instead consider an expected cost formulation. Thus, instead of optimizing for the worst case noise realization, we consider control policies which optimize the given cost function in expectation over possible noise realizations. To do this, we define the following objective function with the following Bellman equation and cost function $C(\cdot, \cdot)$:

$$J^{\pi^j}(x_0^j) = \mathbb{E}_{w_0^j}\left[C(x_0^j, \pi^j(x_0^j)) + J^{\pi^j}(f(x_0^j, u_0^j, w_0^j))\right] \tag{2.2}$$

However, we would like to only consider policies that are robustly constraint-satisfying for all timesteps. Thus, the goal of the control design is to solve the following infinite time optimal control problem:

$$\begin{aligned} J_{0\to\infty}^{j,*}(x_0^j) = \min_{\pi^j(.)} \quad & J^{\pi^j}(x_0^j) \\ \text{s.t.} \quad & x_{t+1}^j = f(x_t^j, u_t^j, w_t^j) \\ & u_t^j = \pi^j(x_t^j) \\ & x_t^j \in \mathcal{X}, u_t^j \in \mathcal{U} \\ & \forall w_t^j \in \mathcal{W}, t \in \{0, 1, \dots\} \end{aligned} \tag{2.3}$$

In this chapter, we present a strategy to iteratively design a feedback policy $\pi^j(.) : \mathcal{F}_\mathcal{G}^j \subseteq \mathcal{X} \to \mathcal{U}$, where $\mathcal{F}_\mathcal{G}^j$ is the domain of π^j for goal set $\mathcal{G}$ (and also the set of allowable initial conditions). Conditioned on the goal set $\mathcal{G}$, the policy design provides guarantees for (i) robust satisfaction of state and input constraints, (ii) convergence in probability of the closed-loop system to $\mathcal{G}$, (iii) iterative improvement: for any $x_0^j = x_0^l$ where $j < l$, expected trajectory cost is non-increasing ($J^{\pi^j}(x_0^j) \geq J^{\pi^{j+1}}(x_0^{j+1})$), and (iv) exploration: the domain of the control policy does not shrink over iterations ($\mathcal{F}_\mathcal{G}^j \subseteq \mathcal{F}_\mathcal{G}^{j+1}$ for all goal sets $\mathcal{G}$ sampled up to iteration j). In Section 2.4.3, we describe how to transfer to a new goal set $\mathcal{H}$ by reusing data from prior iterations while maintaining the same properties.

We adopt the following definitions and assumptions:

Assumption 2.3.1. ***Costs:*** *We consider costs which are zero within the goal set $\mathcal{G}$ and greater than some $\epsilon > 0$ outside the goal set: $\exists \epsilon > 0$ s.t. $C(x, u) \geq \epsilon \mathbb{1}_{\mathcal{G}^C}(x)$ where $\mathbb{1}$ is an indicator function and $\mathcal{G}^C$ is the complement of $\mathcal{G}$.*

Definition 2.3.1. ***Robust Control Invariant Set:*** *A set $\mathcal{A} \subseteq \mathcal{X}$ is a robust control invariant set with respect to dynamics $f(x,u,w)$ and policy class Π, if for an initial condition $x_0 \in \mathcal{X}$ there exists a policy $\pi \in \Pi$ that keep the evolution of the system within $\mathcal{A}$, i.e.,*

$$\forall x \in \mathcal{A},\ \exists \pi \in \Pi \ s.t.\ f(x,\pi(x),w) \in \mathcal{A}, \pi(x) \in \mathcal{U},\ \forall w \in \mathcal{W}.$$

Assumption 2.3.2. ***Robust Control Invariant Goal Set:*** *$\mathcal{G} \subseteq \mathcal{X}$ is a robust control invariant set with respect to the dynamics and the set of state feedback policies Π.*

2.4 Preliminaries

Here we formalize the notion of safe sets, value functions, and how they can be conditioned on specific goals. We also review standard definitions and assumptions.

2.4.1 Safe Set

We first recall the definition of a robust reachable set as in [54]:

Definition 2.4.1. ***Robust Reachable Set:*** *The robust reachable set $\mathcal{R}_t^\pi(x_0^j)$ contains the set of states reachable in t-steps by the system (2.1) in closed loop with π at iteration j:*

$$\begin{aligned}&\mathcal{R}_{t+1}^\pi(x_0^j) = \\ &\big\{x_{t+1} \in \mathbb{R}^n \big|\ \exists w_t \in \mathcal{W},\ x_t \in \mathcal{R}_t^\pi(x_0^j),\ x_{t+1} = f(x_t, \pi(x_t), w_t)\big\}\end{aligned} \tag{2.4}$$

where $\mathcal{R}_0^\pi(x_0^j) = x_0^j$. We define $\mathcal{R}_{t+1}^\pi$ similarly when the input is a set and for time-varying policies.

Now, we define the safe set at iteration j for the goal set $\mathcal{G}$ as in [16].

Definition 2.4.2. ***Safe Set:*** *The safe set $\mathcal{SS}_\mathcal{G}^j$ contains the full evolution of the system at iteration j,*

$$\mathcal{SS}_\mathcal{G}^j = \left\{ \bigcup_{t=0}^{\infty} \mathcal{R}_t^{\pi^j}(x_0^j) \ \bigcup \mathcal{G} \right\}. \tag{2.5}$$

Note that (2.5) is robust control invariant by construction [16]. We could set $\mathcal{SS}_\mathcal{G}^0 = \mathcal{G}$ or initialize the algorithm with a nominal policy π^0. This enables the algorithm to naturally incorporate demonstrations to speed up training.

2.4.2 Value Function

In this section we define the value function over the safe set. In contrast to standard model-based RL strategies [40], our strategy *i)* does not approximate the value function over the entire state space and *ii)* computes a value function associated with the best performing policy from a particular state $x \in \mathcal{SS}^j_{\mathcal{G}}$. These characteristics will be useful to approximate the value function using historical data which are collected over a subset of the state space and for different control policies.

Next, we introduce the expected cost associated with the policy π^j.

Definition 2.4.3. ***Expected Cost:*** *The expected cost of π^j from start state x^j_0 is defined as*

$$J^{\pi^j}(x^j_0) = \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x^j_t, \pi^j(x^j_t))\right] \tag{2.6}$$

Observe that this is the solution to the Bellman equations in Equation 2.2.

Definition 2.4.4. ***Value Function:*** *Recursively define the value function of π^j in closed-loop with* (2.3) *as:*

$$L^{\pi^j}_{\mathcal{G}}(x) = \begin{cases} \mathop{\mathbb{E}}_{w}\left[C(x, \pi^j(x)) + L^{\pi^j}_{\mathcal{G}}(f(x, \pi^j(x), w))\right] & x \in \mathcal{SS}^j_{\mathcal{G}} \\ +\infty & x \notin \mathcal{SS}^j_{\mathcal{G}} \end{cases} \tag{2.7}$$

Let

$$V^{\pi^j}_{\mathcal{G}}(x) = \min_{k \in \{0, \dots j\}} L^{\pi^k}_{\mathcal{G}}(x) \tag{2.8}$$

which is the expected cost-to-go of the best performing prior policy at x.

Observe that $L^{\pi^j}_{\mathcal{G}}$ is defined only on $\mathcal{SS}^j_{\mathcal{G}}$, and $J^{\pi^j} = L^{\pi^j}_{\mathcal{G}}$ on $\mathcal{SS}^j_{\mathcal{G}}$. In the event a nominal policy π^0 is used, we require the following assumption on the initial safe set $\mathcal{SS}^0_{\mathcal{G}}$, which is implicitly a restriction on π^0 for start state x^0_0.

Assumption 2.4.1. ***Safe Set Initial Condition:*** *If a nominal policy π^0 is used, then $\forall x \in \mathcal{SS}^0_{\mathcal{G}},\ L^{\pi^0}_{\mathcal{G}}(x) < \infty$.*

This assumption requires that the nominal policy is able to robustly satisfy constraints and converge in probability to $\mathcal{G}$. If no nominal policy is used, then this assumption is not required. In that case, we let $\mathcal{SS}^0_{\mathcal{G}} = \mathcal{G}$ and $L^{\pi^0}_{\mathcal{G}}(x) = 0\ \forall x \in \mathcal{SS}^0_{\mathcal{G}}$.

2.4.3 Transfer to Novel Goal Sets

While Rosolia et al. [16] studies tasks with fixed goal sets, here we show how the safe set and value function can be modified to transfer the learned policy at iteration $j+1$ to a new robust control invariant goal set $\mathcal{H}$ and reuse data from the earlier iterations to accelerate learning.

Definition 2.4.5. ***Goal Conditioned Safe Set:*** *Define the goal conditioned safe set by collecting the prefixes of all robust reachable sets until they robustly fall in $\mathcal{H}$ as follows:*

$$\mathcal{SS}^j_{\mathcal{H}} = \begin{cases} \bigcup_{k=0}^{k^*} \mathcal{R}_k^{\pi^j} \bigcup \mathcal{H} & \max_{k\in\mathbb{N}} \mathbb{1}\{\mathcal{R}_k^{\pi^j}(x_0^j) \subseteq \mathcal{H}\} = 1 \\ \mathcal{H} & otherwise \end{cases} \tag{2.9}$$

where $k^* = \arg\max_{k\in\mathbb{N}} \mathbb{1}\{\mathcal{R}_k^{\pi^j}(x_0^j) \subseteq \mathcal{H}\}$

We also redefine the value function as follows:

Definition 2.4.6. ***Goal Conditioned Value Function:*** *Recursively define the goal-conditioned value function $L^{\pi^j}_{\mathcal{H}}(x)$ of π^j in closed-loop with (2.3) as:*

$$\begin{cases} \mathbb{E}_w\left[C(x, \pi^j(x)) + L^{\pi^j}(f(x, \pi^j(x), w))\right] & x \in \mathcal{SS}^j_{\mathcal{H}} \setminus \mathcal{H} \\ 0 & x \in \mathcal{H} \\ +\infty & x \notin \mathcal{SS}^j_{\mathcal{H}} \end{cases} \tag{2.10}$$

Define $V^{\pi^j}_{\mathcal{H}}(x) = \min_{k\in\{0,\dots j\}} L^{\pi^k}_{\mathcal{H}}(x)$ *as before.*

This new value function is for a policy that executes π^j but switches to a policy that keeps the system in $\mathcal{H}$ upon entry.

2.5 Policy Design

Here we describe the policy design for optimizing the task cost function while satisfying state and input constraints (Section 2.5.1), and discuss how this can be extended to iteratively expand the policy domain (Section 2.5.2). We consider a fixed goal set $\mathcal{G}$ for clarity, but note that the same formulation holds for other goal sets if the safe set and value function are appropriately defined as in Definitions 2.4.5 and 2.4.6. See Figure 2.1 for an illustration of the full ABC-LMPC policy optimization procedure.

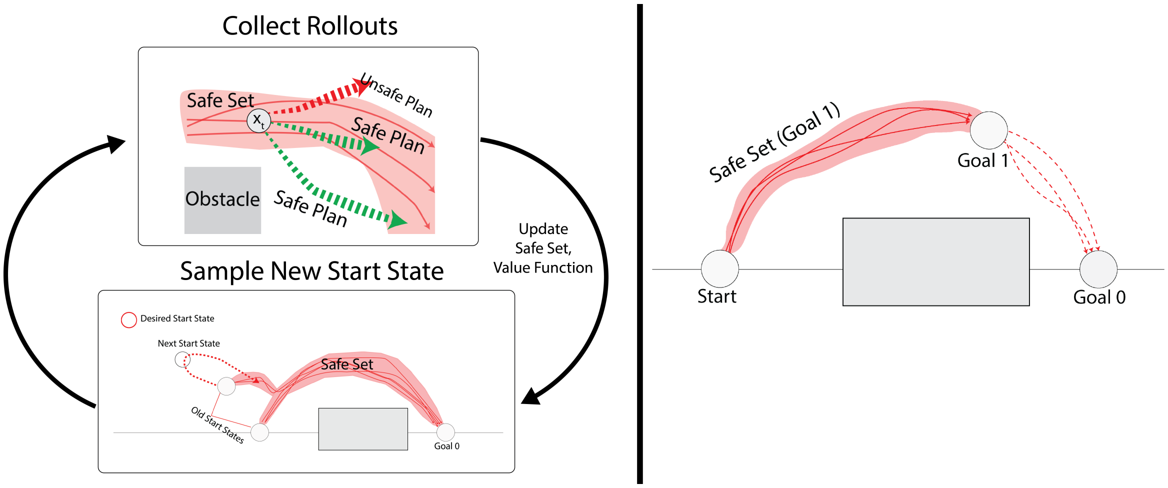


Figure 2.1: **ABC-LMPC Iterative Algorithm (Left):** ABC-LMPC alternates between (1) collecting rollouts under the current policy π^j given $\mathcal{SS}^j_{\mathcal{G}_0}$ and $L^{\pi^j}_{\mathcal{G}_0}$ (by optimizing (2.11)), (2) updating $\mathcal{SS}^{j+1}_{\mathcal{G}_0}$ and $L^{\pi^{j+1}}_{\mathcal{G}_0}$ given the new rollouts, and (3) expanding the policy domain towards a desired start state (by optimizing (2.14)); **Goal Set Transfer (Right):** When a new goal set $\mathcal{G}_1$ is supplied, trajectories to goal $\mathcal{G}_0$ can be reused to estimate a new safe set for a new goal $\mathcal{G}_1$ ($\mathcal{SS}^j_{\mathcal{G}_1}$) and associated value function ($L^{\pi^j}_{\mathcal{G}_1}$).

2.5.1 Task Driven Optimization

At time t of iteration j with goal set $\mathcal{G}$, the policy solves the following receding-horizon trajectory optimization problem with planning horizon $H > 0$:

$$\begin{aligned}
&J^j_{t\to t+H}(x^j_t)\\
&\quad = \min_{\pi_{t:t+H-1|t}} \quad \mathbb{E}_{w^j_{t:t+H-1}}\left[\sum_{i=0}^{H-1} C(x^j_{t+i|t}, u^j_{t+i|t}) + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H|t})\right]\\
&\qquad\quad \text{s.t.} \quad x^j_{t+i+1|t} = f(x^j_{t+i|t}, u^j_{t+i|t}, w_{t+i})\\
&\qquad\qquad\qquad u^j_{t+i|t} = \pi_{t+i|t}(x^j_{t+i|t})\\
&\qquad\qquad\qquad x^j_{t+H|t} \in \bigcup_{k=0}^{j-1} \mathcal{SS}^k_{\mathcal{G}},\\
&\qquad\qquad\qquad x^j_{t:t+H|t} \in \mathcal{X}^{H+1}, \ \ \pi_{t:t+H-1|t} \in \Pi^H\\
&\qquad\qquad\qquad \forall w^j_{t:t+H-1} \in \mathcal{W}^H, \forall i \in \{0, \dots, H-1\}
\end{aligned} \tag{2.11}$$

where $\pi_{t+i|t}$ is the i-th policy in the planning horizon conditioned on x^j_t and $\pi_{t:t+H-1|t} = \{\pi_{t|t}, \dots, \pi_{t+H-1|t}\}$ (likewise for other optimization variables). Let the minimizer of (2.11) be $\pi^{*,j}_{t:t+H-1|t}$. Then, execute the first policy at x^j_t:

$$u^j_t = \pi^j(x^j_t) = \pi^{*,j}_{t|t}(x^j_t) \tag{2.12}$$

Solving 2.11 is typically intractable in practice, so we discuss practical approximations we make to the designed algorithm in Section 10.2.

2.5.2 Start State Expansion

We now describe the control strategy for expanding the policy domain. If there exists a policy π for which the H-step robust reachable set for the start states sampled at iteration j is contained within the current safe set for goal set $\mathcal{G}$, then we can define the feasible set/domain for the policy at iteration j. The domain of π^j for $\mathcal{G}$ is computed by collecting the set of all states for which there exists a sequence of policies which robustly keep the system in $\bigcup_{k=0}^{j-1} \mathcal{SS}_{\mathcal{G}}^k$. Precisely, we define the policy domain as follows:

$$\mathcal{F}_{\mathcal{G}}^j = \{x \mid \exists \pi_{0:H-1} \in \Pi^H \text{ s.t. } \mathcal{R}_H^{\pi_{0:H-1}}(x) \subseteq \bigcup_{k=0}^{j-1} \mathcal{SS}_{\mathcal{G}}^k\} \tag{2.13}$$

This set defines the states from which the system can robustly plan back to $\bigcup_{k=0}^{j-1} \mathcal{SS}_{\mathcal{G}}^k$. Note that the policy domain is a function of the goal set $\mathcal{G}$. While any start state sampled from $\mathcal{F}_{\mathcal{G}}^j$ will ensure feasibility and convergence for goal set $\mathcal{G}$ (proven in Section 10.1), we present a method to compute states from $\mathcal{F}_{\mathcal{G}}^j \setminus \bigcup_{k=0}^{j-1} \mathcal{SS}_{\mathcal{G}}^k$ to expand $\mathcal{F}_{\mathcal{G}}^j$ towards a desired start state, which may not be added to the domain through task-directed exploration. Computing this set is intractable for general nonlinear stochastic systems, so we introduce the following method to approximate this.

At the end of iteration j, we sample a start state $x_S^j \in \bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k$ and seek to execute a sequence of H' exploration policies $\pi_{E,0:H'-1}^j$ which carry the system outside of $\bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k$ and then robustly back into $\bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k$, for all noise realizations $w_{0:H'-2} \in \mathcal{W}^{H'-1}$ where $H' \geq 0$. The sequence of policies $\pi_{E,0:H'-1}^j$ is computed by solving an H'-step optimization problem with a cost function $C_E^j(x,u)$ that encourages exploration outside of $\bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k$ while enforcing that the policy terminates in some state $x_{H'}^j \in \bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k$. In Section 10.2, we discuss some possibilities for $C_E^j(x,u)$, implement one instantiation, and demonstrate that it enables adaptation to novel start states while maintaining policy feasibility. The sequence of policies can computed by solving the following 1-step trajectory optimization problem with $x_0^j = x_S^j$:

$$\begin{aligned}
\pi_{E,0:H'-1}^j(x_S^j) = \underset{\pi_{0:H'-1} \in \Pi^{H'}}{\operatorname{argmin}} \quad & \mathop{\mathbb{E}}_{w_{0:H'-2}^j} \left[\sum_{i=0}^{H'-1} C_E^j(x_i^j, \pi_i(x_i^j)) \right] \\
\text{s.t.} \quad & x_{i+1}^j = f(x_i^j, \pi_i(x_i^j), w_i),\ \forall i \in \{0, \ldots, H'-1\} \\
& x_{H'}^j \in \bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k,\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \\
& x_{0:H'}^j \in \mathcal{X}^{H'+1},\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1}
\end{aligned} \tag{2.14}$$

Let

$$\mathcal{M} = \left(\mathcal{R}_i^{\pi^j_{E,0:H'-1}}(x^j_S)\right)_{i=0}^{H'},$$

be the set of all states reachable in H' steps by π_E and let

$$\mathcal{M}_H = \left(\mathcal{R}_i^{\pi^j_{E,0:H'-1}}(x^j_S)\right)_{i=\max(H'-H,\ 0)}^{H'}.$$

Note that $\forall x \in \mathcal{M}_H$, the policy initialized at x can be robustly guided to $\bigcup_{k=0}^{j} \mathcal{SS}^k_{\mathcal{G}}$ in H steps. At iteration $j+1$, feasible start states can be sampled from $\mathcal{M}_H$ to guide the policy's domain toward a desired target start state. An MPC policy π^j_E could be executed instead to generate these future start states. We could also use the exploration policy to explicitly augment the value function $L^{\pi^j}_{\mathcal{G}}$ and safe set $\mathcal{SS}^j_{\mathcal{G}}$ and thus $\mathcal{F}^j_{\mathcal{G}}$. This could be used for general domain expansion instead of directed expansion towards a desired start state.

Chapter 3

SAVED: Safe RL Using Prior Successes

In this chapter, we relax several of the assumptions made in the previous chapter to adapt ABC-LMPC to the reinforcement learning setting. In particular, because dynamics are unknown, we must identify them from data and leverage uncertainty estimates to satisfy MPC problem constraints, which are now transformed into chance constraints. Additionally, we continuously approximate the safe set in the prior chapter from samples and kernel density estimation.

3.1 Introduction

To use RL in the real world, algorithms need to be efficient, easy to use, and safe, motivating methods which are reliable even with significant dynamical uncertainty. Deep model-based reinforcement learning (deep MBRL) is of significant interest because of its sample efficiency advantages over model-free methods in a variety of tasks, such as assembly, locomotion, and manipulation [25, 27, 28, 55–58]. However, past work in deep MBRL typically requires dense hand-engineered cost functions, which are hard to design and can lead to unintended behavior [59]. It would be easier to simply specify task completion in the cost function, but this setting is challenging due to the lack of expressive supervision. This motivates using demonstrations, which allow the user to roughly specify desired behavior without extensive engineering effort. However, providing high-performing trajectories of the task may be challenging, motivating methods that can rapidly improve upon suboptimal demonstrations that may be supplied via a PID controller or kinesthetically. Furthermore, in many robotic tasks, specifically in domains such as surgery, safe exploration is critical to ensure that the robot does not damage itself or cause harm to its surroundings. To enable this, deep MBRL algorithms must be able to satisfy user-specified (and possibly nonconvex) state-space constraints.

We develop a method to efficiently use deep MBRL in dynamically uncertain environ-

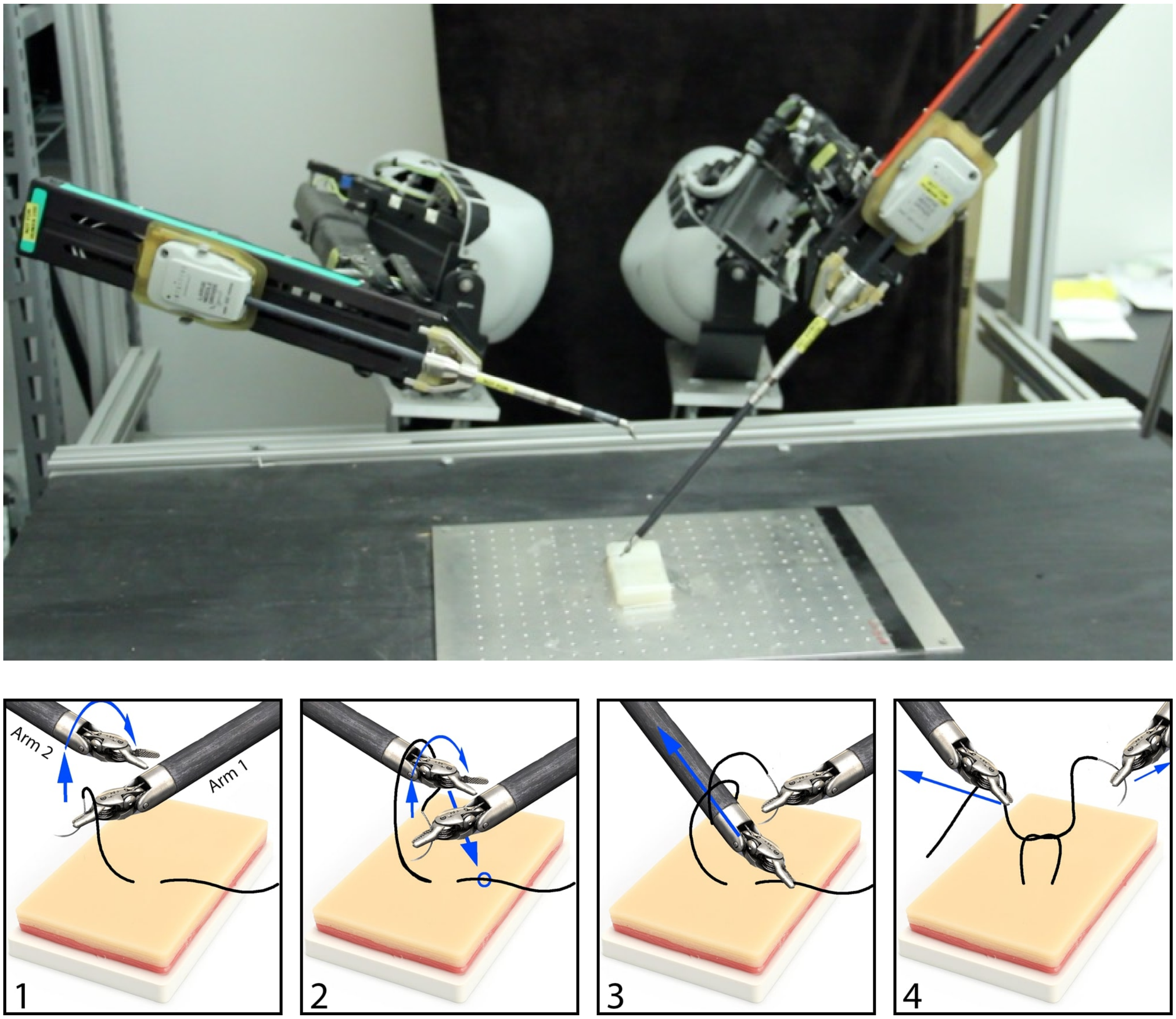


Figure 3.1: SAVED is able to safely learn maneuvers on the da Vinci surgical robot, which is difficult to precisely control [60]. We demonstrate that SAVED is able to optimize inefficient human demonstrations of a surgical knot-tying task, substantially improving on demonstration performance with just 15 training iterations.

ments with both sparse costs and complex constraints. We address the difficulty of hand-engineering cost functions by using a small number of suboptimal demonstrations to provide a signal about task progress in sparse cost environments, which is updated based on agent experience. Then, to enable stable policy improvement and constraint satisfaction, we impose two probabilistic constraints to (1) constrain exploration by ensuring that the agent can plan back to regions in which it is confident in task completion and (2) leverage uncertainty estimates in the learned dynamics to implement chance constraints [61] during learning. The probabilistic implementation of constraints makes this approach broadly applicable, since it can handle settings with significant dynamical uncertainty, where enforcing constraints exactly is difficult.

We introduce a new algorithm motivated by deep model predictive control (MPC) and robust control, Safety Augmented Value Estimation from Demonstrations (SAVED), which enables efficient learning for sparse cost tasks given a small number of suboptimal demonstrations while satisfying the provided constraints. We specifically consider tasks with a tight start state distribution and fixed, known goal set. SAVED is evaluated on MDPs with unknown dynamics, which are iteratively estimated from experience, and with a cost function that only indicates task completion. The contributions of this chapter are (1) a novel method for constrained exploration driven by confidence in task completion, (2) a technique for leveraging model uncertainty to probabilistically enforce complex constraints, enabling obstacle avoidance or optimizing demonstration trajectories while maintaining desired properties, (3) experimental evaluation against 3 state-of-the-art model-free and model-based RL baselines on 8 different environments, including simulated experiments and physical maneuvers on the da Vinci surgical robot. Results suggest that SAVED achieves superior sample efficiency, success rate, and constraint satisfaction rate across all domains considered and can be applied efficiently and safely for learning directly on a real robot.

3.2 Related work

There is significant interest in model-based planning and deep MBRL [25, 40, 55–58] due to the improvements in sample efficiency when planning over learned dynamics compared to model-free methods for continuous control [62, 63]. However, most prior deep MBRL algorithms use hand-engineered dense cost functions to guide exploration and planning, which we avoid by using demonstrations to provide signal about delayed costs. Demonstrations have been leveraged to accelerate learning for a variety of model-free RL algorithms, such as Deep Q Learning [64] and DDPG [65, 66], but model-free methods are typically less sample efficient and cannot anticipate constraint violations since they learn reactive policies [67]. Demonstrations have also been leveraged in model-based algorithms, such as in motion planning with known dynamics [68] and for seeding a learned dynamics model for fast online adaptation using iLQR and a dense cost [58], distinct from the task completion based costs we consider. Unlike traditional motion planning algorithms, which generate open-loop plans to a goal configuration when dynamics are known, we consider designing a closed-loop controller that operates in stochastic dynamical systems where the system dynamics are initially unknown and iteratively estimated from data. Finally, Brown et al. [69] use inverse RL to significantly outperform suboptimal demonstrations, but do not enforce constraints or consistent task completion during learning.

In iterative learning control (ILC), the controller tracks a predefined reference trajectory and data from each iteration is used to improve closed-loop performance [70]. Rosolia et al. [15, 16, 71] provide a reference-free algorithm to iteratively improve the performance of an initial trajectory by using a safe set and terminal cost to ensure recursive feasibility, stability, and local optimality given a known, deterministic nonlinear system or stochastic linear system under certain regularity assumptions. In contrast to Rosolia et al. [15, 16, 71], we consider designing a similar controller in stochastic non-linear dynamical systems where the dynamics are unknown and iteratively estimated from experience. Thus, SAVED uses function approximation to estimate a dynamics model, value function, and safe set. There has also been significant interest in safe RL [72], typically focusing on exploration while satisfying a set of explicit constraints [5, 73, 74], satisfying specific stability criteria [75], or formulating planning via a risk sensitive Markov Decision Process [76, 77]. Distinct from prior work in safe RL and control, SAVED can be successfully applied in settings with both uncertain dynamics and sparse costs by using probabilistic constraints to constrain exploration to feasible regions during learning.

3.3 SAVED: Safety Augmented Value Estimation from Demonstrations

This section describes how SAVED uses a set of suboptimal demonstrations to constrain exploration while satisfying user-specified state space constraints. First, we discuss how SAVED learns system dynamics and a value function to guide learning in sparse cost environments. Then, we motivate and discuss the method used to enforce constraints under uncertainty to both ensure task completion during learning and satisfy user-specified state space constraints.

3.3.1 Assumptions and Preliminaries

In this chapter, we consider stochastic, unknown dynamical systems with a cost function that only identifies task completion. We outline the framework for MBRL using a standard Markov Decision Process formulation. A finite-horizon Markov Decision Process (MDP) is a tuple $(\mathcal{X},\mathcal{U}, P(\cdot,\cdot), T, C(\cdot,\cdot))$ where $\mathcal{X}$ is the feasible (constraint-satisfying) state space and $\mathcal{U}$ is the control space. The stochastic dynamics model P maps a state and control input to a probability distribution over states, T is the task horizon, and C is the cost function. A stochastic control policy π maps an input state to a distribution over $\mathcal{U}$.

We assume that (1) tasks are iterative in nature, and have a fixed low-variance start state distribution and fixed, known goal set $\mathcal{G}$. This is common in a variety of repetitive tasks, such as assembly, surgical knot-tying, and suturing. We further assume that (2) the user specifies an indicator function $\mathbb{1}\left(x \in \mathcal{X}\right)$, which checks whether a state x is constraint-satisfying. Finally, we assume that (3) a modest set of suboptimal but constraint satisfying demos are available, for example from imprecise human teleoperation or a hand-tuned PID controller. This enables rough specification of desired behavior without having to design a dense cost function, allowing us to consider cost functions which only identify task completion: $C(x, u) = \mathbb{1}_{\mathcal{G}^C}(x)$, where $\mathcal{G} \subset \mathcal{X}$ defines a goal set in the state space and $\mathcal{G}^C$ is its complement. We define task success by convergence to $\mathcal{G}$ at the end of the task horizon without violating constraints. Note that under this definition of costs, the problem we consider is equivalent to the shortest time control problem in optimal control, but with initially unknown system dynamics which are iteratively estimated from experience. The applicability of SAVED extends beyond this particular choice of cost function, but we focus on this class due to its convenience and notorious difficulty for reinforcement learning algorithms [66].

Finally, we define the notion of a safe set to enable constrained policy improvement, which is described further in Section 3.3.3. Recent MPC literature [15] motivates constraining exploration to regions in which the agent is confident in task completion, which gives rise to desirable theoretical properties when dynamics are known and satisfy certain regularity conditions [15, 16, 71]. For a trajectory at iteration k, given by $x^k = \{x_t^k | t \in \mathbb{N}\}$, we define the *sampled safe set* as

$$\mathcal{SS}^j = \bigcup_{k \in \mathcal{M}^j} x^k \tag{3.1}$$

where $\mathcal{M}^j = \{k \in [0, j) : \lim_{t \to \infty} x_t^k \in \mathcal{G}\}$ is the set of indices of all successful trajectories before iteration j as in Rosolia et al. [15]. $\mathcal{SS}^j$ contains the states from all iterations before j from which the agent controlled the system to $\mathcal{G}$ and is initialized from demonstrations. A key operating principle of SAVED is to use $\mathcal{SS}^j$ to guide exploration by ensuring that there always exists a way to plan back into a continuous approximation of $\mathcal{SS}^j$. This allows for policy improvement while ensuring that the agent can always return to a state from which it has previously completed the task, enabling consistent task completion during learning.

3.3.2 Algorithm Overview

Deep Model Predictive Control

SAVED uses MPC to optimize costs over a sequence of controls at each state. However, when using MPC, since the current control is computed by solving a finite-horizon approximation to the infinite-horizon control problem, an agent may take shortsighted controls which may make it impossible to complete the task safely, such as planning the trajectory of a race car over a short horizon without considering an upcoming curve [54]. Additionally, the planner receives no feedback or information about task progress when using the indicator task functions used in this chapter. Thus, to guide exploration in temporally-extended tasks, we solve the problem in equation 3.2a, which includes a learned value function in the

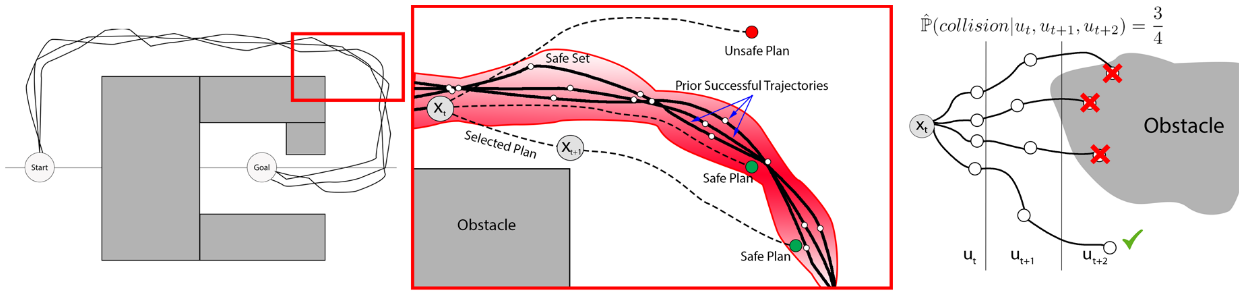


Figure 3.2: **Task Completion Driven Exploration (left):** A density model is used to represent the region in state space where the agent has high confidence in task completion; trajectory samples over the learned dynamics that do not have sufficient density at the end of the planning horizon are discarded. The agent may explore outside the safe set as long as a plan exists to guide the agent back to the safe set from the current state; **Chance Constraint Enforcement (right):** Implemented by sampling imagined rollouts over the learned dynamics for the same sequence of controls multiple times and estimating the probability of constraint violation by the percentage of rollouts that violate a constraint.

objective. Note that $\mathcal{U}^H$ refers to the set of H length control sequences while $\mathcal{X}^{H+1}$ refers to the set of $H+1$ length state sequences. This corresponds to the standard objective in MPC with an appended value function V_ϕ^π, which provides a terminal cost estimate for the current policy at the end of the planning horizon.

While prior work in deep MBRL [25, 55] has primarily focused on planning over learned dynamics, we introduce a learned value function, which is initialized from demonstrations to provide initial signal, to guide planning even in sparse cost settings. The learned dynamics model f_θ and value function V_ϕ^π are each represented with a finite probabilistic ensemble of n neural networks (in this chapter we pick $n = 5$), as is used to represent system dynamics in Chua et al. [25]. The probabilistic ensemble consists of a set of neural networks, each of which output the parameters of a conditional axis-aligned Gaussian distribution and are trained on bootstrapped samples from the training dataset using a maximum likelihood objective as in [25]. Each conditional Gaussian is used to model aleatoric uncertainty in the dynamics, while the bootstrapped ensemble of these models captures epistemic uncertainty due to data availability in different regions of the MDP. SAVED uses the learned stochasticity of the models to enforce probabilistic constraints when planning under uncertainty. These functions are initialized from demonstrations and updated from data collected from each training iteration. See supplementary material for further details on how these networks are trained.

Probabilistic Constraints

The core novelties of SAVED are the additional probabilistic constraints in 3.2c to encourage task completion driven exploration and enforce user-specified chance constraints. First, a non-parametric density model ρ is trained on $\mathcal{SS}^j$, which includes states from prior successful trajectories, including those from demonstrations. ρ constraints exploration by requiring x_{t+H} to fall in a region with high probability of task completion. This enforces cost-driven constrained exploration and iterative improvement, enabling reliable performance even with sparse costs. Note that the agent can still explore new regions, as long as it has a plan that can take it back to the safe set with high probability. Second, we require all elements of $x_{t:t+H}$ to fall in the feasible region $\mathcal{X}^{H+1}$ with probability at least β, which enables probabilistic enforcement of state space constraints. In Section 3.3.3, we discuss the methods used for task completion driven exploration and in Section 3.3.4, we discuss how probabilistic constraints are enforced during learning.

In summary, SAVED solves the following optimization problem at each timestep based on the current state of the system, x_t, which is measured from observations:

$$u^*_{t:t+H-1} = \underset{u_{t:t+H-1}\in\mathcal{U}^H}{\operatorname{argmin}} \mathbb{E}_{x_{t:t+H}}\left[\sum_{i=0}^{H-1} C(x_{t+i}, u_{t+i}) + V^\pi_\phi(x_{t+H})\right] \tag{3.2a}$$

$$\text{s.t. } x_{t+i+1} \sim f_\theta(x_{t+i}, u_{t+i}) \; \forall i \in \{0, \ldots, H-1\} \tag{3.2b}$$

$$\rho_\alpha(x_{t+H}) > \delta, \mathbb{P}\left(x_{t:t+H} \in \mathcal{X}^{H+1}\right) \geq \beta \tag{3.2c}$$

Algorithm 1 SAVED: Safety Augmented Value Estimation from Demonstrations

```
**Require:** Replay Buffer $\mathcal{R}$; value function $V^\pi_\phi(x)$, dynamics model $\hat{f}_\theta(x'|x,u)$, and safety density model $\rho_\alpha(x)$ all seeded with demos; kernel and chance constraint parameters $\alpha$ and $\beta$.
for $i \in \{1, \ldots, N\}$ do
    Sample $x_0$ from start state distribution
    for $t \in \{1, \ldots, T-1\}$ do
        Pick $u^*_{t:t+H-1}$ by solving eq. 3.2 using CEM
        Execute $u^*_t$ and observe $x_{t+1}$
        $\mathcal{R} = \mathcal{R} \cup \{(x_t, u^*_t, C(x_t, u^*_t), x_{t+1})\}$
    end for
    if $x_T \in \mathcal{G}$ then
        Update safety density model $\rho_\alpha$ with $x_{0:T}$
    end if
    Optimize $\theta$ and $\phi$ with $\mathcal{R}$
end for
```

3.3.3 Task Completion Driven Exploration

Under certain regularity assumptions, if states at the end of the MPC planning horizon are constrained to fall in the sampled safe set $\mathcal{SS}^j$, iterative improvement, controller feasibility, and convergence are guaranteed given known stochastic linear dynamics or deterministic nonlinear dynamics [15, 16, 71]. The way we constrain exploration in SAVED builds on this prior work, but we note that unlike Rosolia et al. [15, 16, 71], SAVED is designed for settings in which dynamics are completely unknown, nonlinear, and stochastic. As illustrated in Figure 3.2, the mechanism for constraining exploration allows the agent to generate trajectories that leave the safe set as long as a plan exists to navigate back in, enabling policy improvement. By adding newly successful trajectories to the safe set, the agent is able to further improve its performance. Note that since the safety density model and value function are updated on-policy, the support of the safety density model expands over iterations, while the value function is updated to reflect the current policy. This enables SAVED to improve upon the performance of the demonstrations since on each iteration, it simply needs to be able to plan back to the high support region of a safety density model fit on states from which SAVED was able to complete the task from *all prior iterations* rather than just those visited by the demonstrations.

Since $\mathcal{SS}^j$ is a discrete set, we introduce a continuous approximation by fitting a density model ρ to $\mathcal{SS}^j$. Instead of requiring that $x_{t+H} \in \mathcal{SS}^j$, SAVED instead enforces that $\rho_\alpha(x_{t+H}) > \delta$, where α is a kernel width parameter (constraint 3.2c). Since the tasks considered in this chapter have sufficiently low (< 17) state space dimension, kernel density estimation provides a reasonable approximation. We implement a top-hat kernel density model using a nearest neighbors classifier with a tuned kernel width α and use $\delta = 0$ for all experiments. Thus, all states within Euclidean distance α from the closest state in $\mathcal{SS}^j$ are considered safe under ρ_α, representing states in which the agent is confident in task completion. As the policy improves, it may forget how to complete the task from very old states in $\mathcal{SS}^j$, so such states are evicted from $\mathcal{SS}^j$ to reflect the current policy when fitting ρ_α. We discuss how these constraints are implemented in Section 3.3.4, with further details in the supplementary material. In future work, we will investigate implicit density estimation techniques to scale to high-dimensional settings.

3.3.4 Chance Constraint Enforcement

SAVED leverages uncertainty estimates in the learned dynamics to enforce probabilistic constraints on its trajectories. This allows SAVED to handle complex, user-specified state space constraints to avoid obstacles or maintain certain properties of demonstrations without a user-shaped or time-varying cost function. During MPC trajectory optimization, control sequences are sampled from a truncated Gaussian distribution that is iteratively updated using the cross-entropy method (CEM) [25]. Each control sequence is simulated multiple times over the stochastic dynamics model as in [25] and the average return of the simulations is used to score the sequence. However, unlike Chua et al. [25], we implement chance constraints by discarding control sequences if more than $100 \cdot (1 - \beta)\%$ of the simulations violate constraints (constraint 3.2c), where β is a user-specified tolerance. Note that the β parameter controls the tradeoff between ensuring sufficient exploration to learn the dynamics and satisfying specified constraints. This is illustrated in Figure 3.2. The task completion constraint (Section 3.3.3) is implemented similarly, with control sequences discarded if any of the simulated rollouts do not terminate in a state with sufficient density under ρ_α.

3.3.5 Algorithm Pseudocode

We summarize SAVED in Algorithm 1. The dynamics, value function, and state density model are initialized from suboptimal demonstrations. At each iteration, we sample a start state and then controls are generated by solving equation 3.2 using the cross-entropy method (CEM) at each timestep. Transitions are collected in a replay buffer to update the dynamics, value function, and safety density model at the end of each iteration. The state density model is only updated if the last trajectory was successful.

3.4 Experiments

We evaluate SAVED on simulated continuous control benchmarks and on real robotic tasks with the da Vinci Research Kit (dVRK) [78] against state-of-the-art deep RL algorithms and find that SAVED outperforms all baselines in terms of sample efficiency, success rate, and constraint satisfaction rate during learning. All tasks use $C(x, u) = \mathbb{1}_{\mathcal{G}^C}(x)$ (Section 3.3.1), which yields a controller which maximizes the time spent inside the goal set. All algorithms are given the same demonstrations and are evaluated by measuring iteration cost, success rate, and constraint satisfaction rate (if applicable). Tasks are only considered successfully completed if the agent reaches and stays in $\mathcal{G}$ until the end of the episode. Constraint violation results in early termination of the episode.

For all experiments, we run each algorithm 3 times to control for stochasticity in training and plot the mean iteration cost vs. time with error bars indicating the standard deviation over the 3 runs. Additionally, when reporting task success rate and constraint satisfaction rate, we show bar plots indicating the median value over the 3 runs with error bars between the lowest and highest value over the 3 runs. When reporting the iteration cost of SAVED and all baselines, any constraint violating trajectory is reported by assigning it the maximum possible iteration cost T, where T is the task horizon. Thus, any constraint violation is treated as a catastrophic failure. We plan to explore soft constraints as well in future work. Furthermore, for all simulated tasks, we also report best achieved iteration costs, success rates, and constraint satisfaction rates for model-free methods after 10,000 iterations since they take much longer to start performing the task even when supplied with demonstrations.

For SAVED, dynamics models and value functions are each represented with a probabilistic ensemble of 5, 3 layer neural networks with 500 hidden units per layer with swish activations as used in Chua et al. [25]. To plan over the dynamics, the TS-∞ trajectory sampling method from [25] is used. We use 5 and 30 training epochs for dynamics and value function training when initializing from demonstrations. When updating the models after each training iteration, 5 and 15 epochs are used for the dynamics and value functions respectively. Value function initialization is done by training the value function using the true cost-to-go estimates from demonstrations. However, when updated on-policy, the value function is trained using temporal difference error (TD-1) on a replay buffer containing prior states. The safety density model, ρ, is trained on a fixed history of states from which the agent was able to reach the goal (safe states), where this history can be tuned based on the experiment (see supplement). We represent the density model using kernel density estimation with a tophat kernel. Instead of modifying δ for each environment, we set $\delta = 0$ (keeping points with positive density), and modify α (the kernel parameter/width), which works well in practice. See the supplementary material for additional experiments, videos, and ablations with respect to choice of α, β, and demonstration quantity/quality. We also include further details on baselines, network architectures, hyperparameters, and training procedures.

3.4.1 Baselines

We consider the following set of model-free and model-based baseline algorithms. To enforce constraints for model-based baselines, we augment the algorithms with the simulation based method described in Section 3.3.4. Because model-free baselines have no such mechanism to readily enforce constraints, we instead apply a very large cost when constraints are violated. See supplementary material for an ablation of the reward function used for model-free baselines.

1. **Behavior Cloning (Clone)**: Supervised learning on demonstration trajectories.
2. **PETS from Demonstrations (PETSfD)**: Probabilistic ensemble trajectory sampling (PETS) from Chua et al [25] with the dynamics model initialized with demo trajectories and planning horizon long enough to plan to the goal (judged by best performance of SAVED).
3. **PETSfD Dense**: PETSfD with tuned dense cost.
4. **Soft Actor Critic from Demonstrations (SACfD)**: Model-free RL algorithm, Soft Actor Critic [62], where demo transitions are used for training initially.
5. **Overcoming Exploration in Reinforcement Learning from Demonstrations (OEFD)**: Model-free algorithm from Nair et al. [66] which combines model-free RL with a behavior cloning loss on the demonstrations to accelerate learning.
6. **SAVED (No SS)**: SAVED without the *sampled safe set* constraint described in Section 3.3.3.

3.4.2 Simulated Navigation

To evaluate whether SAVED can efficiently and safely learn temporally extended tasks with nonconvex constraints, we consider a 4-dimensional (x, y, v_x, v_y) navigation task in which a point mass is navigating to a goal set, which is a unit ball centered at the origin. The agent can exert force in cardinal directions and experiences drag coefficient ψ and Gaussian process noise $z_t \sim \mathcal{N}(0, \sigma^2 I)$ in the dynamics. We use $\psi = 0.2$ and $\sigma = 0.05$ in all experiments in

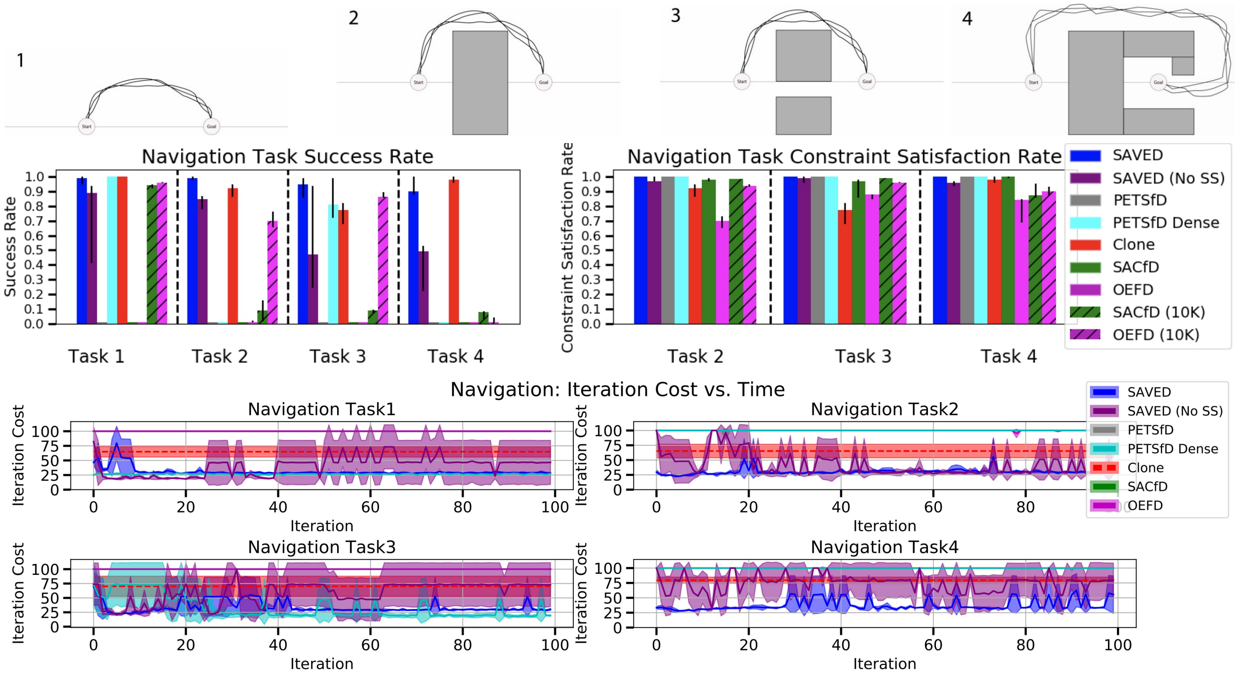


Figure 3.3: **Navigation Domains:** SAVED is evaluated on 4 navigation tasks. Tasks 2-4 contain obstacles, and task 3 contains a channel for passage to $\mathcal{G}$ near the x-axis. SAVED learns significantly faster than all RL baselines on tasks 2 and 4. In tasks 1 and 3, SAVED has lower iteration cost than baselines using sparse costs, but does worse than PETSfD Dense, which is given dense Euclidean norm costs to find the shortest path to the goal. For each task and algorithm, we report success and constraint satisfaction rates over the first 100 training iterations and also over the first 10,000 iterations for SACfD and OEFD. We observe that SAVED has higher success and constraint satisfaction rates than other RL algorithms using sparse costs across all tasks, and even achieves higher rates in the first 100 training iterations than model-free algorithms over the first 10,000 iterations.

this domain. Demonstrations trajectories are generated by guiding the robot along a very suboptimal hand-tuned trajectory for the first half of the trajectory before running LQR on a quadratic approximation of the true cost. Gaussian noise is added to the demonstrator policy. Additionally, we use a planning horizon of 15 for SAVED and 25, 30, 30, 35 for PETSfD for tasks 1-4 respectively. The 4 experiments run on this environment are:

1. **Long navigation task to the origin:** $x_0 = (-100, 0)$ We use 50 demonstrations with average return of 73.9 and kernel width $\alpha = 3$.
2. **Large obstacle blocking the x-axis:** This environment is difficult for approaches that use a Euclidean norm cost function due to local minima. We use 50 demonstrations with average return of 67.9, kernel width $\alpha = 3$, and chance constraint parameter $\beta = 1$.
3. **Large obstacle with a small channel near the x-axis:** This environment is difficult for the algorithm to optimally solve since the iterative improvement of paths

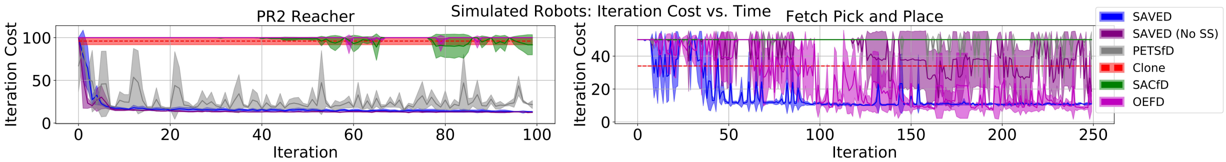


Figure 3.4: **Simulated Robot Experiments Performance:** SAVED achieves better performance than all baselines on both tasks. We use 20 demonstrations with average iteration cost of 94.6 for the reacher task and 100 demonstrations with average iteration cost of 34.4 for the pick and place task. For the reacher task, the safe set constraint does not improve performance, likely because the task is very simple, but for pick and place, we see that the safe set constraint adds significant training stability.

taken by the agent is constrained. We use $x_0 = (-50, 0)$, 50 demonstrations with average return of 67.9, kernel width $\alpha = 3$, and chance constraint parameter $\beta = 1$.

4. **Large obstacle surrounds the goal set with a small channel for entry:** This environment is very difficult to solve without demonstrations. We use $x_0 = (-50, 0)$, 100 demonstrations with average return of 78.3, kernel width $\alpha = 3$, and chance constraint parameter $\beta = 1$.

SAVED has a higher success rate than all other RL baselines using sparse costs, even including model-free baselines over the first 10,000 iterations, while never violating constraints across all navigation tasks. Furthermore, this performance advantage is amplified with task difficulty. Only Clone and PETSfD Dense ever achieve a higher success rate, but Clone does not improve upon demonstration performance (Figure 3.3) and PETSfD Dense has additional information about the task. Furthermore, SAVED learns significantly more efficiently than all RL baselines on all navigation tasks except for tasks 1 and 3, in which PETSfD Dense with a Euclidean norm cost function finds a better solution. While SAVED (No SS) can complete the tasks, it has a much lower success rate than SAVED, especially in environments with obstacles as expected, demonstrating the importance of the *sampled safe set* constraint. Note that SACfD, OEFD, and PETSfD make essentially no progress in the first 100 iterations and never complete any of the tasks in this time, although they mostly satisfy constraints. After 10,000 iterations of training, SACfD and OEFD achieve average best iteration costs of 23.7 and 23.8 respectively on task 1, 21 and 21.7 respectively on task 2, 17.3 and 19 respectively on task 3, and 23.7 and 40 respectively on task 4. Thus, we see that SAVED achieves comparable performance in the first 100 iterations to the asymptotic performance of model-free RL algorithms while maintaining consistent task completion and constraint satisfaction during learning.

3.4.3 Simulated Robot Experiments

To evaluate whether SAVED also outperforms baselines on standard unconstrained environments, we consider sparse versions of two common simulated robot tasks: the torque-controlled PR2 Reacher environment from Chua et al. [25] with a fixed goal and on a pick

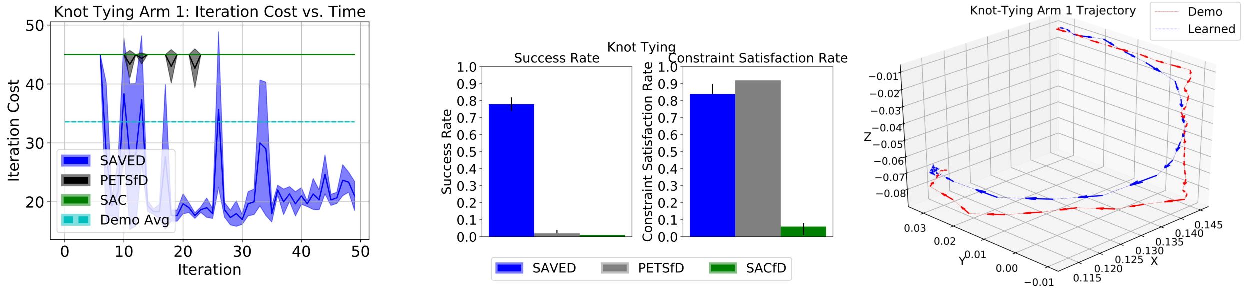


Figure 3.5: **Physical Surgical Knot-Tying: Training Performance:** After just 15 iterations, the agent completes the task relatively consistently with only a few failures, and converges to a iteration cost of 22, faster than demos, which have an average iteration cost of 34. In the first 50 iterations, both baselines mostly fail, and are less efficient than demos when they do succeed; **Trajectories:** SAVED quickly learns to speed up with only occasional constraint violations.

and place task with a simulated, position-controlled Fetch robot from [79]. The reacher task involves controlling the end-effector of a simulated PR2 robot to a small ball in $\mathbb{R}^3$. The state representation consists of 7 joint positions, 7 joint velocities, and the goal position. The goal set is specified by a 0.05m radius Euclidean ball in state space. Suboptimal demonstrations are generated with average cost 94.6 by training PETS with a shaped cost function that heavily penalizes large torques. We use $\alpha = 15$ for SAVED and a planning horizon of 25 for both SAVED and PETSfD. SACfD and OEFD achieve a best iteration cost of 9 and 60 respectively over 10,000 iterations of training averaged over the 3 runs. The pick and place task involves picking up a block from a fixed location on a table and also guiding it to a small ball in $\mathbb{R}^3$. The task is simplified by automating the gripper motion, which is difficult for SAVED to learn due to the bimodality of gripper controls, which is hard to capture with the unimodal truncated Gaussian distribution used during CEM sampling. The state representation for the task consists of (end effector relative position to object, object relative position to goal, gripper jaw positions). Suboptimal demonstrations are generated by hand-tuning a controller that slowly but successfully completes the task with average iteration cost 34.4. We use a safe set buffer size of 5000 and $\alpha = 0.05$. We use a planning horizon of 10 for SAVED and 20 for PETSfD. SACfD and OEFD both achieve a best iteration cost of 6 over 10,000 iterations of training averaged over the 3 runs.

SAVED learns faster than all baselines on both tasks (Figure 3.4) and exhibits significantly more stable learning in the first 100 and 250 iterations for the reacher and pick and place tasks respectively. However, while SAVED is substantially more sample efficient than SACfD and OEFD for these tasks, both algorithms achieve superior asymptotic performance.

3.4.4 Physical Robot Experiments

We evaluate the ability of SAVED to learn a surgical knot-tying task with nonconvex state space constraints on the da Vinci Research Kit (dVRK) [78]. The dVRK is cable-driven and has relatively imprecise controls, motivating model learning [60]. Furthermore, safety is paramount due to the cost and delicate structure of the arms. The goal of these tasks is to speed up demonstration trajectories while still maintaining properties of the trajectories that result in a task completion. This is accomplished by constraining learned trajectories to fall within a tight, 1 cm tube of the demos. The goal set is represented with a 1 cm ball in $\mathbb{R}^3$ and the robot is controlled via delta-position control, with a maximum control magnitude of 1 cm during learning for safety. Robot experiments are very time consuming due to interactive data collection, so training RL algorithms on limited physical hardware is difficult without sample efficient algorithms. We include additional experiments on a Figure-8 tracking task in the supplementary material.

Surgical Knot-Tying

SAVED is used to optimize demonstrations of a surgical knot-tying task on the dVRK, using the same multilateral motion as in [80]. Demonstrations are hand-engineered for the task, and then policies are optimized for one arm (arm 1), while a hand-engineered policy is used for the other arm (arm 2). While arm 1 wraps the thread around arm 2, arm 2 simply moves down, grasps the other end of the thread, and pulls it out of the phantom as shown in Figure 3.1. Thus, we only expect significant performance gain by optimizing the policy for the portion of the arm 1 trajectory which involves wrapping the thread around arm 2. We only model the motion of the end-effectors in 3D space. We use $\beta = 0.8$, $\alpha = 0.05$, planning horizon 10, and 100 demonstrations with average cost 34.4 for SAVED. We use a planning horizon of 20 and $\beta = 1.$ for PETSfD. SAVED quickly learns to smooth out demo trajectories, with a success rate of over 75% (Figure 3.5) during training, while baselines are unable to make sufficient progress in this time. PETSfD rarely violates constraints, but also almost never succeeds, while SACfD almost always violates constraints and never completes the task. Training SAVED directly on the real robot for 50 iterations takes only about an hour, making it practical to train on a real robot for tasks where data collection is expensive. At execution-time (post-training), we find that SAVED is very consistent, successfully tying a knot in 20/20 trials with average iteration cost of 21.9 and maximum iteration cost of 25 for the arm 1 learned policy, significantly more efficient than demos which have an average iteration cost of 34. See supplementary material for trajectory plots of the full knot-tying trajectory and the Figure-8 task.

3.5 Discussion and Future Work

We present SAVED, a model-based RL algorithm that can efficiently learn a variety of robotic control tasks in the presence of dynamical uncertainty, sparse cost feedback, and complex constraints by using suboptimal demonstrations to constrain exploration to regions in which the agent is confident in task completion. We then empirically evaluate SAVED on 6 simulated benchmarks and on a knot-tying task on a real surgical robot. Results suggest that SAVED is more sample efficient and has higher success and constraint satisfaction rates than all RL baselines and can be efficiently and safely trained on a real robot. In future work, we will explore convergence and safety guarantees for SAVED and extensions to a wide distribution of start states and goal sets. Additionally, a limitation of SAVED is that solving the MPC objective with CEM makes high frequency control difficult. In future work, we will explore distilling the learned controller into a reactive policy to enable fast policy evaluation in practice.

Chapter 4

LS^3: Scaling Safe RL Using Prior Successes

In this chapter, we will further relax the algorithm in the previous chapter to adapt it to high dimensional observation spaces such as images. Image-based safe sets are challenging for SAVED, because SAVED assumes access to a natural metric to compare the dynamical distance between states. In this chapter, we will discuss techniques for learning safe sets in a latent space as well as latent obstacles and dynamics.

4.1 Introduction

Visual planning over learned forward dynamics models is a popular area of research in robotic control from images [81–87], as it enables closed-loop, model-based control for tasks where the state of the system is not directly observable or difficult to analytically model, such as the configuration of a sheet of fabric or segment of cable. These methods learn predictive models over either images or a learned latent space, which can then be used to plan actions which maximize some task reward. While these approaches have significant promise, there are several open challenges in learning policies from visual observations. First, reward specification is particularly challenging for visuomotor control tasks, because high-dimensional observations often do not expose the necessary features required to design dense, informative reward functions [88], especially for long-horizon tasks. Second, while many prior reinforcement learning methods have been successfully applied to image-based control tasks [46, 89–92], learning policies from image observations often requires extensive exploration due to the high dimensionality of the observation space and the difficulties in reward specification, making safe and efficient learning exceedingly challenging.

One promising strategy for efficiently learning safe control policies is to learn a safe set [15, 93], which captures the set of states from which the agent is known to behave safely, which is often reformulated as the set of states where it has previously completed the task. When used to restrict exploration, this safe set can be used to enable highly efficient and safe learning [14, 15, 24], as exploration is restricted to states in which the agent is confident in task success. However, while these safe sets can give rise to algorithms with a number of appealing theoretical properties such as convergence to a goal set, constraint satisfaction, and iterative improvement [14–16], using them for controller design for practical problems requires developing continuous approximations at the expense of maintaining theoretical guarantees [24]. This choice of continuous approximation is a key element in determining the applications to which these safe sets can be used for control.

Prior works have presented approaches which collect a discrete safe set of states from previously successful trajectories and represent a continuous relaxation of this set by constructing a convex hull of these states [15] or via kernel density estimation with a tophat kernel function [24]. While these approaches have been successful for control tasks with low-dimensional states, extending them to high-dimensional observations presents two key challenges: (1) *scalability:* these prior methods cannot be efficiently applied when the number of observations in prior successful trajectories is large, as querying safe set inclusion scales linearly with number of samples it contains and (2) *representation capacity:* both of these prior approaches do not scale well to high dimensional observations and are limited in the space of continuous sets that they can efficiently represent. Applying these ideas to visuomotor control is even more challenging, since images do not directly expose details about the system state or dynamics that are typically needed for formal controller analysis [12, 14, 15].

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

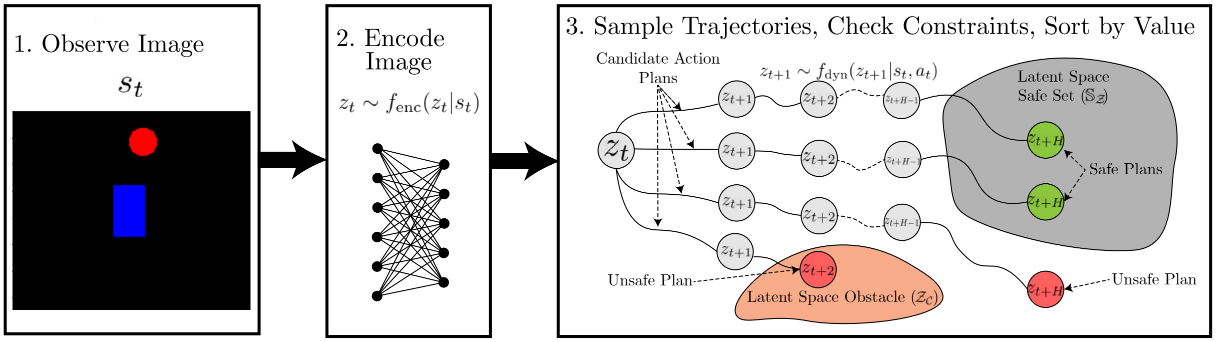

1. Observe Image
s_t
2. Encode Image
$z_t \sim f_{\text{enc}}(z_t|s_t)$
3. Sample Trajectories, Check Constraints, Sort by Value
Candidate Action Plans
$z_{t+1} \sim f_{\text{dyn}}(z_{t+1}|s_t, a_t)$
z_t
z_{t+1}
z_{t+2}
z_{t+H-1}
z_{t+H}
Latent Space Safe Set ($\mathbb{S}_{\mathcal{Z}}$)
Safe Plans
Unsafe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

Figure 4.1: **Latent Space Safe Sets (LS^3):** At time t, LS^3 observes an image s_t of the environment. The image is first encoded to a latent vector $z_t \sim f_{\text{enc}}(z_t|s_t)$. Then, LS^3 uses a sampling-based optimization procedure to optimize H-length action sequences by sampling H-length latent trajectories over the learned latent dynamics model f_{dyn}. For each sampled trajectory, LS^3 checks whether latent space obstacles are avoided and if the terminal state in the trajectory falls in the latent space safe set. The terminal state constraint encourages the algorithm to maintain plans back to regions of safety and task confidence, but still enables exploration. For feasible trajectories, the sum of rewards and value of the terminal state are computed and used for sorting. LS^3 executes the first action in the optimized plan and then performs this procedure again at the next timestep.

4.2 Related Work

4.2.1 Safe, Iterative Learning Control

In iterative learning control (ILC), the agent tracks a reference trajectory and uses data from controller rollouts to refine tracking performance [70]. Rosolia et al. [15, 16, 71] present a new class of algorithms, known as Learning Model Predictive Control (LMPC), which are reference-free and instead iteratively *improve* upon the performance of an initial feasible trajectory. To achieve this, Rosolia et al. [15, 16, 71] use data from controller rollouts to learn a safe set and value function, with which recursive feasibility, stability, and local optimality can be guaranteed given a known, deterministic nonlinear system or stochastic linear system under certain regularity assumptions. However, a core challenge with these algorithms is that they assume known system dynamics, and cannot be applied to high-dimensional control problems. Thananjeyan et al. [24] extends the LMPC framework to higher dimensional settings in which system dynamics are unknown and must be learned, but the visuomotor control setting introduces a number of new challenges as learned system dynamics, safe sets, and value functions must flexibly scale to visual inputs. Richards et al. [93] designs expressive safe sets for fixed policies using neural network classifiers with Lyapunov constraints. In contrast, LS^3 constructs a safe set for an improving policy by optimizing a task cost function instead of uniformly expanding across the state space.

4.2.2 Model Based Reinforcement Learning

There has been significant recent progress in algorithms which combine ideas from model-based planning and control with deep learning [25, 40, 55–58]. These algorithms are gaining popularity in the robotics community as they enable leaning complex policies from data while maintaining some of the sample efficiency and safety benefits of classical model-based control techniques. However, these algorithms typically require hand-engineered dense cost functions for task specification, which can often be difficult to provide, especially in high-dimensional spaces. This motivates leveraging demonstrations (possibly suboptimal) to provide an initial signal regarding desirable agent behavior. There has been some prior work on leveraging demonstrations in model-based algorithms such as Quinlan et al. [68] and Ichnowski et al. [94], which use model-based control with known dynamics to refine initially suboptimal motion plans, and Fu et al. [58], which uses demonstrations to seed a learned dynamics model for fast online adaptation using iLQR [58]. Thananjeyan et al. [24] and Zhu et al. [95] present ILC algorithms which rapidly improve upon suboptimal demonstrations when system dynamics are unknown. However, these algorithms either require knowledge of system dynamics [68, 94] or are limited to low-dimensional state spaces [24, 58, 95] and cannot be flexibly applied to visuomotor control tasks.

4.2.3 Reinforcement Learning from Pixels

Reinforcement learning and model-based planning from visual observations is gaining significant recent interest as RGB images provide an easily available observation space for robot learning [81, 96]. Recent work has proposed a number of model-free and model-based algorithms that have seen success in laboratory settings in a number of robotic tasks when learning from visual observations [46, 81, 91, 92, 96–100]. However, two core issues that prevent application of many RL algorithms in practice, inefficient exploration and safety, are significantly exacerbated when learning from high-dimensional visual observations in which the space of possible behaviors is very large and the features required to determine whether the robot is safe are not readily exposed. There has been significant prior work on addressing inefficiencies in exploration for visuomotor control such as latent space planning [82, 96, 100] and goal-conditioned reinforcement learning [46, 92]. However, safe reinforcement learning for visuomotor tasks has received substantially less attention. Thananjeyan et al. [8] and Kahn et al. [101] present reinforcement learning algorithms which estimate the likelihood of constraint violations to avoid them [8] or reduce the robot's velocity [101]. Unlike these algorithms, which focus on presenting methods to avoid violating user-specified constraints, LS^3 additionally provides consistent task completion during learning by limiting exploration to the neighborhood of prior task successes. This difference makes LS^3 less susceptible to the challenges of unconstrained exploration present in standard model-free reinforcement learning algorithms.

4.3 Problem Statement

We consider an agent interacting in a finite horizon goal-conditioned Markov Decision Processes (MDP) which can be described with the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, P(\cdot|\cdot,\cdot), R(\cdot,\cdot), \mu, T)$. $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ maps a state and action to a probability distribution over subsequent states, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, μ is the initial state distribution ($s_0 \sim \mu$), and T is the time horizon. In this chapter, the agent is only provided with RGB image observations $s_t \in \mathbb{R}_+^{W \times H \times 3} = \mathcal{S}$, where W and H are the image width and height in pixels, respectively. We consider iterative tasks, where the agent must reach a fixed goal set $\mathcal{G} \subseteq \mathcal{S}$ as efficiently as possible and the support of μ is small. While there are a number of possible choices of reward functions that would encourage fast convergence to $\mathcal{G}$, providing shaped reward functions can be exceedingly challenging, especially when learning from high dimensional observations. Thus, as in Thananjeyan et al. [24], we consider a sparse reward function that only indicates task completion: $R(s, a, s') = 0$ if $s' \in \mathcal{G}$ and -1 otherwise. To incorporate constraints, we augment $\mathcal{M}$ with an extra constraint indicator function $\mathcal{C} : \mathcal{S} \to \{0, 1\}$ which indicates whether a state satisfies user-specified state-space constraints, such as avoiding known obstacles. This is consistent with the modified CMDP formulation used in [8]. We assume that R and $\mathcal{C}$ can be evaluated on the current state of the system, but may be approximated using prior data for use during planning. We make this assumption because in practice we plan over predicted future states, which may not be predicted at sufficiently high fidelity to expose the necessary information to directly evaluate R and $\mathcal{C}$ during planning.

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

$$\pi^* = \arg\max_{\pi \in \Pi} \{R^\pi : \mathbb{E}_{s_0 \sim \mu}\left[P_C^\pi(s_0)\right] \leq \delta_\mathcal{C}\} \tag{4.1}$$

We assume that the agent is provided with an offline dataset $\mathcal{D}$ of transitions in the environment of which some subset $\mathcal{D}_{\text{constraint}} \subsetneq \mathcal{D}$ are constraint violating and some subset $\mathcal{D}_{\text{success}} \subsetneq \mathcal{D}$ appear in successful demonstrations from a suboptimal supervisor. As in [8], $\mathcal{D}_{\text{constraint}}$ contains examples of constraint violating behaviors (for example from prior runs of different policies or collected under human supervision) so that the agent can learn about states which violate user-specified constraints.

4.4 Latent Space Safe Sets (LS^3)

We describe how LS^3 uses demonstrations and online interaction to safely learn iteratively improving policies. Section 4.4.1 describes how we learn a low-dimensional latent representation of image observations to facilitate efficient model-based planning. To enable this

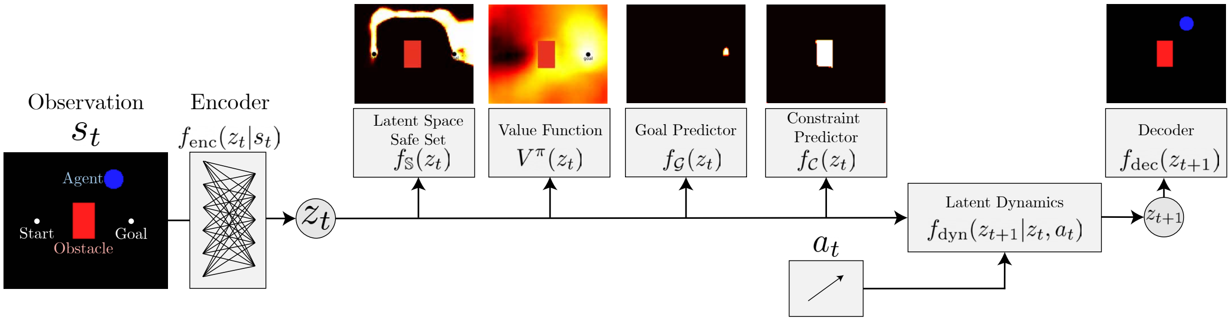


Figure 4.2: **LS^3 Learned Models**: LS^3 learns a low-dimensional latent representation of image-observations (Section 4.4.1) and learns a dynamics model, value function, reward function, constraint classifier, and safe set for constrained planning and task-completion driven exploration in this learned latent space. These models are then used for model-based planning to maximize the total value of predicted latent states (Section 4.4.3) while enforcing the safe set (Section 4.4.2) and user-specified constraints (Section 4.4.3).

planning, we learn a probabilistic forward dynamics model as in [25] in the learned latent space and models to estimate whether plans will likely complete the task (Section 4.4.2) and to estimate future rewards and constraint violations (Section 4.4.3) from predicted trajectories. In Section 4.4.4, we discuss how these components are synthesized in LS^3. Dataset $\mathcal{D}$ is expanded using online rollouts of LS^3 and used to update all latent space models (Sections 4.4.2 and 4.4.3) after every K rollouts. See Algorithm 2 and the supplement for further details on training procedures and data collection.

Algorithm 2 Latent Space Safe Sets (LS^3)

```
Require: offline dataset $\mathcal{D}$, number of updates $U$
Train VAE encoder $f_{\text{enc}}$ and decoder $f_{\text{dec}}$ (Section 4.4.1) using data from $\mathcal{D}$
Train dynamics $f_{\text{dyn}}$, safe set classifier $f_{\mathbb{S}}$(Section 4.4.2), and the value function $V$ goal
indicator $f_{\mathcal{G}}$, and constraint estimator $f_{\mathcal{C}}$ (Section 4.4.3) using data from $\mathcal{D}$.
for $j \in \{1, \ldots, U\}$ do
    for $k \in \{1, \ldots, K\}$ do
        Sample starting state $s_0$ from $\mu$.
        for $t \in \{1, \ldots, T\}$ do
            Choose and execute $a_t$ (Section 4.4.4)
            Observe $s_{t+1}$, reward $r_t$, constraint $c_t$.
            $\mathcal{D} := \mathcal{D} \cup \{(s_t, a_t, s_{t+1}, r_t, c_t)\}$
        end for
    end for
    Update $f_{\text{dyn}}$, $V$, $f_{\mathcal{G}}$, $f_{\mathcal{C}}$, and $f_{\mathbb{S}}$ with data from $\mathcal{D}$.
end for
```

4.4.1 Learning a Latent Space for Planning

Learning compressed representations of images has been a popular approach in vision based control to facilitate efficient algorithms for planning and control which can reason about lower dimensional inputs [82, 86, 96, 100, 102, 103]. To learn such a representation, we train a β-variational autoencoder [104] on states in $\mathcal{D}$ to map states to a probability distribution over a d-dimensional latent space $\mathcal{Z}$. The resulting encoder network $f_{\text{enc}}(z|s)$ is then used to sample latent vectors $z_t \sim f_{\text{enc}}(z_t|s_t)$ to train a forward dynamics model, value function, reward estimator, constraint classifier, safe set, and combine these elements to define a policy for model-based planning. Motivated by Laskin et al. [105], during training we augment inputs to the encoder with random cropping, which we found to be helpful in learning representations that are useful for planning. For all environments we use a latent dimension of $d = 32$, as in [82] and found that varying d did not significantly affect performance.

4.4.2 Latent Safe Sets for Model-Based Control

LS^3 learns a binary classifier for latent states to learn a latent space safe set that represents states from which the agent has high confidence in task completion based on prior experience. Because the agent can reach the goal from these states, they are safe: the agent can avoid constraint violations by simply completing the task as before. While classical algorithms use known dynamics to construct safe sets, we approximate this set using successful trajectories from prior iterations. At each iteration j, the algorithm collects K trajectories in the environment. We then define the sampled safe set at iteration j, $\mathbb{S}^j$, as the set of states from which the agent has successfully navigated to $\mathcal{G}$ in iterations 0 through j of training, where demonstrations trajectories are those collected at iteration 0. We refer to the dataset collecting all these states as $\mathcal{D}_{\text{success}}$. This discrete set is difficult to plan to with continuous-valued state distributions so we leverage data from $\mathcal{D}_{\text{success}}$ (data in the sampled safe set), data from $\mathcal{D} \setminus \mathcal{D}_{\text{success}}$ (data outside the sampled safe set), and the learned encoder from Section 4.4.1 to learn a continuous relaxation of this set in latent space (the latent safe set). We train a neural network with a binary cross-entropy loss to learn a binary classifier $f_{\mathbb{S}}(\cdot)$ that predicts the probability of a state s_t with encoding z_t being in $\mathbb{S}^j$. To mitigate the negative bias that appears when trajectories that start in safe regions fail, we utilize the intuition that if a state $s_{t+1} \in \mathbb{S}^j$ then it is likely that s_t is also safe. To do this, rather than just predict $\mathbb{1}_{\mathbb{S}^j}$, we train $f_{\mathbb{S}}$ with a recursive objective to predict $\max(\mathbb{1}_{\mathbb{S}^j}, \gamma_{\mathbb{S}} f_{\mathbb{S}}(s_{t+1}))$. The relaxed latent safe set is parameterized by the superlevel sets of $f_{\mathbb{S}}$, where the level $\delta_{\mathbb{S}}$ is adaptively set during execution: $\mathbb{S}^j_{\mathcal{Z}} = \{z_t | f_{\mathbb{S}}(\cdot)(z_t) \geq \delta_{\mathbb{S}}\}$.

4.4.3 Reward and Constraint Estimation

In this chapter, we define rewards based on whether the agent has reached a state $s \in \mathcal{G}$, but we need rewards that are defined on predictions from the dynamics, which may not correspond to valid real images. To address this, we train a classifier $f_{\mathcal{G}} : \mathcal{Z} \to \{0, 1\}$ to map the encoding of a state to whether the state is contained in $\mathcal{G}$ using terminal states in $\mathcal{D}_{\text{success}}$ (which are known to be in $\mathcal{G}$) and other states in $\mathcal{D}$. However, in the temporally-extended, sparse reward tasks we consider, reward prediction alone is insufficient because rewards only indicate whether the agent is in the goal set, and thus provide no signal on task progress unless the agent can plan to the goal set. To address this, as in prior MPC-literature [14, 15, 24, 88], we train a recursively-defined value function (details in the supplement). Similar to the reward function, we use the encoder (Section 4.4.1) to train a classifier $f_{\mathcal{C}} : \mathcal{Z} \to [0, 1]$ with data of constraint violating states from $\mathcal{D}_{\text{constraint}}$ and the constraint satisfying states in $\mathcal{D} \setminus \mathcal{D}_{\text{constraint}}$ to map the encoding of a state to the probability of constraint violation.

4.4.4 Model-Based Planning with LS^3

LS^3 aims to maximize total rewards attained in the environment while limiting constraint violation probability within some threshold $\delta_{\mathcal{C}}$ (equation 4.1). We optimize an approximation of this objective over an H-step receding horizon with model-predictive control. Precisely, LS^3 solves the following optimization problem to generate an action to execute at timestep t:

$$\begin{aligned}
\underset{a_{t:t+H-1} \in \mathcal{A}^H}{\arg\max} \quad & \mathbb{E}_{z_{t:t+H}} \left[\sum_{i=1}^{H-1} f_{\mathcal{G}}(z_{t+i}) + V^{\pi}(z_{t+H}) \right] && (4.2) \\
\text{s.t.} \quad & z_t \sim f_{\text{enc}}(z_t | s_t) && (4.3) \\
& z_{k+1} \sim f_{\text{dyn}}(z_{k+1} | z_k, a_k) \; \forall k \in \{t, \dots, t+H-1\} && (4.4) \\
& \hat{\mathbb{P}}\left(z_{t+H} \in \mathbb{S}^{j-1}_{\mathcal{Z}} \right) \geq 1 - \delta_{\mathbb{S}} && (4.5) \\
& \hat{\mathbb{P}}(z_{t+i} \in \mathcal{Z}_{\mathcal{C}}) \leq \delta_{\mathcal{C}} \; \forall i \in \{0, \dots, H-1\} && (4.6)
\end{aligned}$$

In this problem, the expectations and probabilities are taken with respect to the learned, probabilistic dynamics model $f_{\text{dyn}}(z_{t+1} | z_t, a_t)$. The optimization problem is solved approximately using the cross-entropy method (CEM) [106] which is a popular optimizer in model-based RL [8, 14, 24, 107, 108].

The objective function is the expected sum of future rewards if the agent executes $a_{t:t+H-1}$ and then subsequently executes π (equation 4.2). First, the current state s_t is encoded to z_t (equation 4.3). Then, for a candidate sequence of actions $a_{t:t+H-1}$, an H-step latent trajectory $\{z_{t+1}, \dots, z_{t+H}\}$ is sampled from the learned dynamics f_{dyn} (equation 4.4). LS^3 constrains exploration using two chance constraints: (1) the terminal latent state in the plan must fall in the safe set (equation 4.5) and (2) all latent states in the trajectory must satisfy user-specified state-space constraints (equation 4.6). $\mathcal{Z}_{\mathcal{C}}$ is the set of all latent states such that the corresponding observation is constraint violating. The optimizer estimates constraint satisfaction probabilities for a candidate action sequence by simulating it repeatedly over f_{dyn}. The first chance constraint ensures the agent maintains the ability to return to safe states where it knows how to do the task within H steps if necessary. Because the agent replans at each timestep, the agent need not return to the safe set: during training, the safe

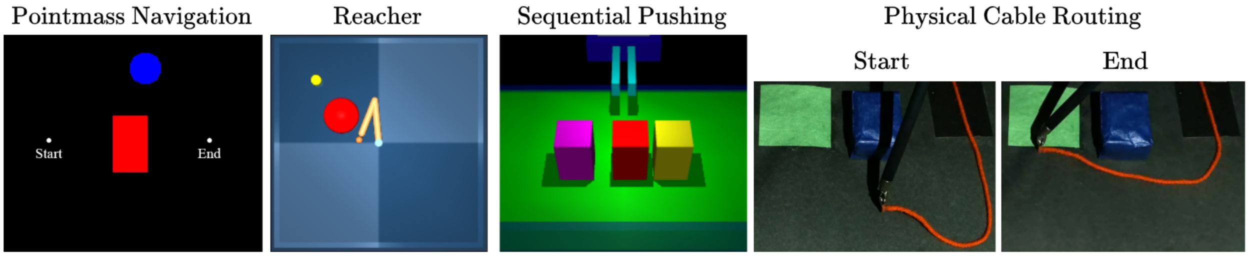


Figure 4.3: **Experimental Domains:** LS^3 is evaluated on 3 long-horizon, image-based, simulation environments: a visual navigation domain where the goal is to navigate the blue point mass to the right goal set while avoiding the red obstacle, a 2 degree of freedom reacher arm where the task is to navigate around a red obstacle to reach the yellow goal set, and a sequential pushing task where the robot must push each of 3 blocks forward a target displacement from left to right. We also evaluate LS^3 on a physical, cable-routing task on a da Vinci Surgical Robot, where the goal is to guide a red cable to a green target without the cable or robot arm colliding with the blue obstacle. This requires learning visual dynamics, because the agent must model how the rest of the cable will deform during manipulation to avoid collisions with the obstacle.

set expands, enabling further exploration. In practice, we set $\delta_{\mathbb{S}}$ for the safe set classifier $f_{\mathcal{S}}$ adaptively as described in the supplement. The second chance constraint encourages constraint violation probability of no more than $\delta_{\mathcal{C}}$. After solving the optimization problem, the agent executes the first action in the plan: $\pi(z_t) = a_t$ where a_t is the first element of $a^*_{t:t+H-1}$, observes a new state, and replans.

4.5 Experiments

We evaluate LS^3 on 3 robotic control tasks in simulation and a physical cable routing task on the da Vinci Research Kit (dVRK) [78]. Safe RL is of particular interest for surgical robots such as the dVRK due to its delicate structure, motivating safety, and relatively imprecise controls [24, 60], motivating closed-loop control. We study whether LS^3 can learn more safely and efficiently than algorithms that do not structure exploration based on prior task successes.

4.5.1 Comparisons

We evaluate LS^3 in comparison to prior algorithms that behavior clone suboptimal demonstrations before exploring online (**SACfD**) [62] or leverage offline reinforcement learning to learn a policy using all offline data before updating the policy online (**AWAC**) [109]. For both of these comparisons we enforce constraints via a tuned reward penalty of λ for constraint violations as in [3]. We also implement a version of SACfD with a learned recovery policy (**SACfD+RRL**) using the Recovery RL algorithm [8] to use prior constraint violating data to try to avoid constraint violating states. We then compare LS^3 to an ablated version without the safe set constraint (just binary classification (BC)) in equation 4.5 (**LS^3 (−Safe**

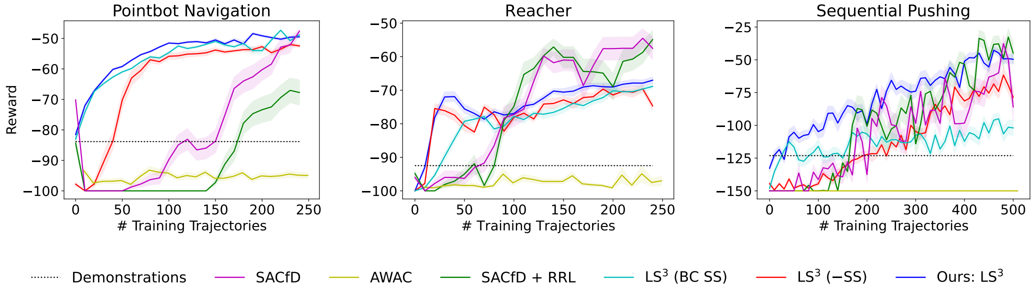


Figure 4.4: **Simulation Experiments Results:** Learning curves showing mean and standard error over 10 random seeds. We see that LS^3 learns more quickly than baselines and ablations. Although SACfD and SACfD+RRL converge to similar reward values, LS^3 is much more sample efficient and stable across random seeds.

Set)) to evaluate if the safe set promotes consistent task completion and stable learning. Finally, we compare LS^3 to an ablated version of the safe set classifier (Section 4.4.2) without a recursive objective, where the classifier is just trained to predict $\mathbb{1}_{\mathbb{S}^j}$ (**LS^3 (BC SS)**). See the supplement for details on hyperparameters and offline data used for LS^3 and prior algorithms.

4.5.2 Evaluation Metrics

For each algorithm on each domain, we aggregate statistics over random seeds (10 for simulation experiments, 3 for the physical experiment), reporting the mean and standard error across the seeds. We present learning curves that show the total sum reward for each training trajectory to study how efficiently LS^3 and the comparisons learn each task. Because all tasks use the sparse task completion based rewards defined in Section 4.3, the total reward for a trajectory is the time to reach the goal set, where more negative rewards correspond to slower convergence to $\mathcal{G}$. Thus, for a task with task horizon T, a total reward greater than $-T$ implies successful task completion. The state is frozen in place upon constraint violation until the task horizon elapses. We also report task success and constraint satisfaction rates for LS^3 and comparisons during learning to study (1) the degree to which task completion influences sample efficiency and (2) how safely different algorithms explore. LS^3 collects $K = 10$ trajectories in between training phases on simulated tasks and $K = 5$ in between training phases for physical tasks, while the SACfD and AWAC comparisons update their parameters after each timestep. This presents a metric in terms of the amount of data collected across algorithms.

4.5.3 Domains

In simulation, we evaluate LS^3 on 3 vision-based continuous control domains that are illustrated in Figure 4.3. We evaluate LS^3 and comparisons on a constrained visual navigation task (Pointmass Navigation) where the agent navigates from a fixed start state to a fixed goal set while avoiding a large central obstacle. We study this domain to gain intuition and visualize the learned value function, goal/constraint indicators, and safe set in Figure 4.2. We then study a constrained image-based reaching task (Reacher) based on [110], where the objective is to navigate the end effector of a 2-link planar robotic arm to a yellow goal position without the end-effector entering a red stay out zone. We then study a challenging sequential image-based robotic pushing domain (Sequential Pushing), in which the objective is to push each of the 3 blocks forward on the table without pushing them to either side and causing them to fall out of the workspace. Finally, we evaluate LS^3 with an image-based physical experiment on the da Vinci Research Kit (dVRK) [111] (Figure 4.3), where the objective is to guide the endpoint of a cable to a goal region without letting the cable or end effector collide with an obstacle. The Pointmass Navigation and Reaching domains have a task horizon of $T = 100$ while the Sequential Pushing domain and physical experiment have task horizons of $T = 150$ and $T = 50$ respectively. See the supplement for more details on all domains.

4.5.4 Simulation Results

We find that LS^3 is able to learn more stably and efficiently than all comparisons across all simulated domains while converging to similar performance within 250 trajectories collected online (Figure 4.4). LS^3 is able to consistently complete the task during learning, while the comparisons, which do not learn a safe set to structure exploration based on prior successes, exhibit much less stable learning. Additionally, in Table 4.1 and Table 4.2, we report the task success rate and constraint violation rate of all algorithms during training. We find that LS^3 achieves a significantly higher task success rate than comparisons on all tasks. We also find that LS^3 violates constraints less often than comparisons on the Reacher task, but violates constraints more often than SACfD and SACfD+RRL on the other domains. This is because SACfD and SACfD+RRL spend much less time in the neighborhood of constraint violating states during training due to their lower task success rates. Because they do not efficiently learn to perform the tasks, they do not violate constraints as often. We find that the AWAC comparison achieves very low task performance. While AWAC is designed for offline reinforcement learning, to the best of our knowledge, it has not been previously evaluated on long-horizon, image-based tasks as in this paper, which we hypothesize are very challenging for it.

We find LS^3 has a lower success rate when the safe set constraint is removed (LS^3(−Safe Set)) as expected. The safe set is particularly important in the sequential pushing task, and LS^3 (−Safe Set) has a much lower task completion rate than LS^3. LS^3 without the recursive classification objective from Section 4.4.2 (LS^3 (BC SS)) has similar performance to LS^3 on the navigation environment, but learns substantially more slowly on the Reacher environment and performs significantly worse than LS^3 on the more challenging Pushing environment as the learned safe set is unable to exploit temporal structure to distinguish safe states from unsafe states. See the supplement for details on experimental parameters and offline data

Table 4.1: **Task Success Rate over all Training Episodes:** We present the mean and standard error of training-time task completion rate over 10 random seeds. We find LS^3 outperforms all comparisons across all 3 domains, with the gap increasing for the challenging sequential pushing task.

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

Table 4.2: **Constraint Violation Rate:** We report mean and standard error of training-time constraint violation rate over 10 random seeds. LS^3 violates constraints less than comparisons on the Reacher task, but SAC and SACfD+RRL achieve lower constraint violation rates on the Navigation and Pushing tasks, likely due to spending less time in the neighborhood of constraint violating regions due to their much lower task success rates.

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

used for LS^3 and comparisons and ablations studying the effect of the planning horizon and threshold used to define the safe set.

4.5.5 Physical Results

In physical experiments, we compare LS^3 to SACfD and SACfD+RRL (Figure 4.5) on the physical cable routing task illustrated in Figure 4.3. We find LS^3 quickly outperforms the suboptimal demonstrations while succeeding at the task significantly more often than both comparisons, which are unable to learn the task and also violate constraints more than LS^3. We hypothesize that the difficulty of reasoning about cable collisions and deformation from images makes it challenging for prior algorithms to make sufficient task progress as they do not use prior successes to structure exploration. See the supplement for details on experimental parameters and offline data used for LS^3 and comparisons.

4.6 Discussion and Future Work

We present LS^3, a scalable algorithm for safe and efficient policy learning for visuomotor tasks. LS^3 structures exploration by learning a safe set in a learned latent space, which captures the set of states from which the agent is confident in task completion. LS^3 then

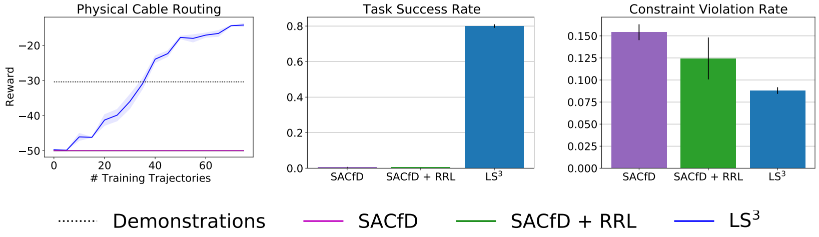


Figure 4.5: **Physical Cable Routing Results:** We present learning curves, task success rates and constraint violation rates with a mean and standard error across 3 random seeds. LS^3 learns a more efficient policy than the demonstrator while still violating constraints less than comparisons, which are unable to learn the task.

ensures that the agent can plan back to states in the safe set, encouraging consistent task completion during learning. Experiments suggest that LS^3 can safely and efficiently learn 4 visuomotor control tasks, including a challenging sequential pushing task in simulation and a cable routing task on a physical robot. In future work, we are excited to explore further physical evaluation of LS^3 on safety critical visuomotor control tasks and applications to systems with dynamic constraints on velocity or acceleration.

Chapter 5

Model-based Learning with a Model-free Policy

The previous chapters discuss how to use model-based policies for safe reinforcement learning. However, the model-based optimizations are very expensive and may be too costly to run at execution time. In this chapter, we will discuss a technique for distilling a model-based policy that is improving over time into a model-free policy that is fast to execute. To theoretically justify this technique, we will discuss a few theoretical results that suggests that training a model-free policy from an improving offline supervisor (the model-based agent) results in sublinear regret.

In robotics there is significant interest in using human or algorithmic supervisors to train policies via imitation learning [112–114]. For example, a trained surgeon with experience teleoperating a surgical robot can provide successful demonstrations of surgical maneuvers [115]. Similarly, known dynamics models can be used by standard control techniques, such as model predictive control (MPC), to generate controls to optimize task reward [116, 117]. However, there are many cases in which the supervisor is not fixed, but is *converging* to improved behavior over time, such as when a human is initially unfamiliar with a teleoperation interface or task or when the dynamics of the system are initially unknown and estimated with experience from the environment when training an algorithmic controller. Furthermore, these supervisors are often *slow*, as humans can struggle to execute stable, high-frequency actions on a robot [117] and model-based control techniques, such as MPC, typically require computationally expensive stochastic optimization techniques to plan over complex dynamics models [25, 55, 118]. This motivates algorithms that can distill supervisors which are both *converging* and *slow* into policies that can be efficiently executed in practice. The idea of distilling improving algorithmic controllers into reactive policies has been explored in a class of reinforcement learning (RL) algorithms known as dual policy iteration (DPI) [119–121], which alternate between optimizing a reactive learner with imitation learning and a model-based supervisor with data from the learner. However, past methods have mostly been applied in discrete settings [119, 120] or make specific structural assumptions on the supervisor [121].

This chapter analyzes learning from a converging supervisor in the context of on-policy imitation learning. Prior analysis of on-policy imitation learning algorithms provide regret guarantees given a fixed supervisor [122–125]. We consider a converging sequence of supervisors and show that similar guarantees hold for the regret against the best policy in hindsight with labels from the converged supervisor, even when only intermediate supervisors provide labels during learning. Since the analysis makes no structural assumptions on the supervisor, this flexibility makes it possible to use any off-policy method as the supervisor in the presented framework, such as an RL algorithm or a human, provided that it converges to a good policy on the learner's distribution. We implement an instantiation of this framework with the deep MPC algorithm PETS [25] as an improving supervisor and maintain the data efficiency of PETS while significantly reducing online computation time, accelerating both policy learning and evaluation.

The key contribution of this chapter is a new framework for on-policy imitation learning from a converging supervisor. We present a new notion of static and dynamic regret in this setting and provide sublinear regret guarantees by showing a reduction from this new notion of regret to the standard notion for the fixed supervisor setting. The dynamic regret result is particularly unintuitive, as it indicates that it is possible to do well on each round of learning compared to a learner with labels from the converged supervisor, even though labels are only provided by intermediate supervisors during learning. We then show that the presented framework relaxes assumptions on the supervisor in DPI and perform simulated continuous control experiments suggesting that when a PETS supervisor [25] is used, we can outperform other deep RL baselines while achieving up to an 80-fold speedup in policy evaluation. Experiments on a physical surgical robot yield up to an 20-fold reduction in query time and 53% reduction in policy evaluation time after accounting for hardware constraints.

5.1 Related Work

On-policy imitation learning algorithms that directly learn reactive policies from a supervisor were popularized with DAgger [122], which iteratively improves the learner by soliciting supervisor feedback on the learner's trajectory distribution. This yields significant performance gains over analogous off-policy methods [126, 127]. On-policy methods have been applied with both human [128] and algorithmic supervisors [117], but with a fixed supervisor as the guiding policy. We propose a setting where the supervisor improves over time, which is common when learning from a human or when distilling a computationally expensive, iteratively improving controller into a policy that can be efficiently executed in practice. Recently, convergence results and guarantees on regret metrics such as dynamic regret have been shown for the fixed supervisor setting [124, 125, 129]. We extend these results and present a static and dynamic analysis of on-policy imitation learning from a convergent sequence of supervisors. Recent work proposes using inverse RL to outperform an improving supervisor [69, 130]. We instead study imitation learning in this context to use an evolving supervisor for policy learning.

Model-based planning has seen significant interest in RL due to the benefits of leveraging structure in settings such as games and robotic control [119–121]. Deep model-based reinforcement learning (MBRL) has demonstrated superior data efficiency compared to model-free methods and state-of-the-art performance on a variety of continuous control tasks [25, 55, 118]. However, these techniques are often too computationally expensive for high-frequency execution, significantly slowing down policy evaluation. To address the online burden of model-based algorithms, Sun et al. [121] define a novel class of algorithms, dual policy iteration (DPI), which alternate between optimizing a fast learner for policy evaluation using labels from a model-based supervisor and optimizing a slower model-based supervisor using trajectories from the learner. However, past work in DPI either involves planning in discrete state spaces [119, 120], or making specific assumptions on the structure of the model-based controller [121]. We discuss how the converging supervisor framework is connected to DPI, but enables a more flexible supervisor specification. We then provide a practical algorithm by using the deep MBRL algorithm PETS [25] as an improving supervisor to achieve fast policy evaluation while maintaining the data efficiency of PETS.

5.2 Converging Supervisor Framework and Preliminaries

5.2.1 On-Policy Imitation Learning

We consider continuous control problems in a finite-horizon Markov decision process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, P(\cdot,\cdot), T, R(\cdot,\cdot))$ where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. The stochastic dynamics model P maps a state s and action a to a probability distribution over states, T is the task horizon, and R is the reward function. A deterministic control policy π maps an input state in $\mathcal{S}$ to an action in $\mathcal{A}$. The goal in RL is to learn a policy π over the MDP which induces a trajectory distribution that maximizes the sum of rewards along the trajectory. In imitation learning, this objective is simplified by instead optimizing a surrogate loss function which measures the discrepancy between the actions chosen by learned parameterized policy π_θ and supervisor ψ.

Rather than directly optimizing R from experience, on-policy imitation learning involves executing a policy in the environment and then soliciting feedback from a supervisor on the visited states. This is in contrast to off-policy methods, such as behavior cloning, in which policy learning is performed entirely on states from the supervisor's trajectory distribution. The surrogate loss of a policy π_θ along a trajectory is a supervised learning cost defined by the supervisor relabeling the trajectory's states with actions. The goal of on-policy imitation is to find the policy minimizing the corresponding surrogate risk on its own trajectory distribution. On-policy algorithms typically adhere to the following iterative procedure: (1) at iteration i, execute the current policy π_{θ_i} by deploying the learner in the MDP, observing states and actions as trajectories; (2) Receive labels for each state from the supervisor ψ; (3) Update π_{θ_i} according to the supervised learning loss to generate $\pi_{\theta_{i+1}}$.

On-policy imitation learning has often been viewed as an instance of online optimization or online learning [122, 124, 125]. Online optimization is posed as a game between an adversary, which generates a loss function l_i at iteration i and an algorithm, which plays a policy π_{θ_i} in an attempt to minimize the total incurred losses. After observing l_i, the algorithm updates its policy $\pi_{\theta_{i+1}}$ for the next iteration. In the context of imitation learning, the loss $l_i(\cdot)$ at iteration i corresponds to the supervised learning loss function under the current policy. The loss function $l_i(\cdot)$ can then be used to update the policy for the next iteration. The benefit of reducing on-policy imitation learning to online optimization is that well-studied analyses and regret metrics from online optimization can be readily applied to understand and improve imitation learning algorithms. Next, we outline a theoretical framework in which to study on-policy imitation learning with a converging supervisor.

5.2.2 Converging Supervisor Framework (CSF)

We begin by presenting a set of definitions for on-policy imitation learning with a converging supervisor in order to analyze the static regret (Section 5.3.1) and dynamic regret (Section 5.3.2) that can be achieved in this setting. In this chapter, we assume that policies π_θ are parameterized by a parameter θ from a convex compact set $\Theta \subset \mathbb{R}^d$ equipped with the l_2-norm, which we denote with $\|\cdot\|$ for simplicity for both vectors and operators.

Definition 5.2.1. ***Supervisor:*** *We can think of a converging supervisor as a sequence of supervisors (labelers), $(\psi_i)_{i=1}^{\infty}$, where ψ_i defines a deterministic controller $\psi_i : \mathcal{S} \to \mathcal{A}$. Supervisor ψ_i provides labels for imitation learning policy updates at iteration i.*

Definition 5.2.2. ***Learner:*** *The learner is represented at iteration i by a parameterized policy $\pi_{\theta_i} : \mathcal{S} \to \mathcal{A}$ where π_{θ_i} is differentiable function in the policy parameter $\theta_i \in \Theta$.*

We denote the state and action at timestep t in the trajectory τ sampled at iteration i by the learner with s_t^i and a_t^i respectively.

Definition 5.2.3. ***Losses:*** *We consider losses at each round i of the form: $l_i(\pi_\theta, \psi_i) = \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T}\|\pi_\theta(s_t^i) - \psi_i(s_t^i)\|^2\right]$ where $p(\tau|\theta_i)$ defines the distribution of trajectories generated by π_{θ_i}. Gradients of l_i with respect to θ are defined as $\nabla_\theta l_i(\pi_{\theta_i}, \psi_i) = \nabla_\theta l_i(\pi_\theta, \psi_i)\big|_{\theta=\theta_i}$.*

For analysis of the converging supervisor setting, we adopt the following standard assumptions. The assumptions in this section and the loss formulation are consistent with those in Hazan [131] and Ross et al. [122] for analysis of online optimization and imitation learning algorithms. The loss incurred by the agent is the population risk of the policy, and extension to empirical risk can be derived via standard concentration inequalities as in Ross et al. [122].

Assumption 5.2.1. ***Strongly convex losses:*** *$\forall \theta_i \in \Theta$, $l_i(\pi_\theta, \psi)$ is strongly convex with respect to θ with parameter $\alpha \in \mathbb{R}^+$. Precisely, we assume that*

$$l_i(\pi_{\theta_2}, \psi) \geq l_i(\pi_{\theta_1}, \psi) + \nabla_\theta l_i(\pi_{\theta_1}, \psi)^T(\theta_2 - \theta_1) + \frac{\alpha}{2}\|\theta_2 - \theta_1\|^2 \quad \forall\ \theta_1, \theta_2 \in \Theta$$

The expectation over $p(\tau|\theta_i)$ in Assumption 5.2.1 preserves strong convexity of the squared loss for an individual sample, which is assumed to be convex in θ.

Assumption 5.2.2. ***Bounded operator norm of policy Jacobian:*** *$\|\nabla_\theta \pi_{\theta_i}(s)\| \leq G$ $\forall s \in \mathcal{S}$, $\ \forall\ \theta, \theta_i \in \Theta$ where $G \in \mathbb{R}^+$.*

Assumption 5.2.3. ***Bounded action space:*** *The action space $\mathcal{A}$ has diameter δ. Equivalently stated: $\delta = \sup_{a_1, a_2 \in \mathcal{A}} \|a_1 - a_2\|$.*

5.3 Regret Analysis

We analyze the performance of well-known algorithms in on-policy imitation learning and online optimization under the converging supervisor framework. In this setting, we emphasize that the goal is to achieve low loss $l_i(\pi_{\theta_i}, \psi_N)$ with respect to labels from the last observed supervisor ψ_N. We achieve these results through regret analysis via reduction of on-policy imitation learning to online optimization, where regret is a standard notion for measuring the performance of algorithms. We consider two forms: static and dynamic regret [132], both of which have been utilized in previous on-policy imitation learning analyses [122, 124]. In this chapter, regret is defined with respect to the expected losses under the trajectory distribution induced by the realized sequence of policies $(\pi_{\theta_i})_{i=1}^N$. Standard concentration inequalities can be used for finite sample analysis as in Ross et al. [122].

Using static regret, we can show a loose upper bound on average performance with respect to the last observed supervisor with minimal assumptions, similar to [122]. Using dynamic regret, we can tighten this upper bound, showing that θ_i is optimal in expectation on its own distribution with respect to ψ_N for certain algorithms, similar to [124, 129]; however, to achieve this stronger result, we require an additional continuity assumption on the dynamics of the system, which was shown to be necessary by Cheng et al. [125]. To harness regret analysis in imitation learning, we seek to show that algorithms achieve *sublinear regret* (whether static or dynamic), denoted by $o(N)$ where N is the number of iterations. That is, the regret should grow at a slower rate than linear in the number of iterations. While existing algorithms can achieve sublinear regret in the fixed supervisor setting, we analyze regret with respect to the last observed supervisor ψ_N, even though the learner is only provided labels from the intermediate ones during learning. See supplementary material for all proofs.

5.3.1 Static Regret

Here we show that as long as the supervisor labels are Cauchy, i.e. if $\forall s \in \mathcal{S}$, $\forall N > i, \|\psi_i(s) - \psi_N(s)\| \leq f_i$ where $\lim_{i\to\infty} f_i = 0$, it is possible to achieve sublinear static regret with respect to the best policy in hindsight with labels from ψ_N for the whole dataset. This is a more difficult metric than is typically considered in regret analysis for on-policy imitation learning since labels are provided by the converging supervisor ψ_i at iteration i, but regret is evaluated with respect to the best policy given labels from ψ_N. Past work has shown that it is possible to obtain sublinear static regret in the fixed supervisor setting under strongly convex losses for standard on-policy imitation learning algorithms such as online gradient descent [131] and DAgger [122]; we extend this and show that the additional asymptotic regret in the converging supervisor setting depends only on the convergence rate of the supervisor. The standard notion of static regret is given in Definition 5.3.1.

Definition 5.3.1. *The static regret with respect to the sequence of supervisors $(\psi_i)_{i=1}^N$ is given by the difference in the performance of policy π_{θ_i} and that of the best policy in hindsight under the average trajectory distribution induced by the incurred losses with labels from current supervisor ψ_i.*

$$\text{Regret}_N^S((\psi_i)_{i=1}^N) = \sum_{i=1}^N l_i(\pi_{\theta_i}, \psi_i) - \sum_{i=1}^N l_i(\pi_{\theta^*}, \psi_i) \ \textit{where}\ \theta^* = \arg\min_{\theta\in\Theta} \sum_{i=1}^N l_i(\pi_\theta, \psi_i)$$

However, we instead analyze the more difficult regret metric presented in Definition 5.3.2 below.

Definition 5.3.2. *The static regret with respect to the supervisor ψ_N is given by the difference in the performance of policy π_{θ_i} and that of the best policy in hindsight under the average trajectory distribution induced by the incurred losses with labels from the last observed supervisor ψ_N.*

$$\text{Regret}_N^S(\psi_N) = \sum_{i=1}^N l_i(\pi_{\theta_i}, \psi_N) - \sum_{i=1}^N l_i(\pi_{\theta^\star}, \psi_N) \ \textit{where}\ \theta^\star = \arg\min_{\theta\in\Theta} \sum_{i=1}^N l_i(\pi_\theta, \psi_N)$$

Theorem 5.3.1. $\text{Regret}_N^S(\psi_N)$ *can be bounded above as follows:*

$$\text{Regret}_N^S(\psi_N) \leq \text{Regret}_N^S((\psi_i)_{i=1}^N) + 4\delta \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^T \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right]$$

Chapter 5.3.1 essentially states that the expected static regret in the converging supervisor setting can be decomposed into two terms: one that is the standard notion of static regret, and an additional term that scales with the rate at which the supervisor changes. Thus, as long as there exists an algorithm to achieve sublinear static regret on the standard problem, the only additional regret comes from the evolution of the supervisor. Prior work has shown that algorithms such as online gradient descent [131] and DAgger [122] achieve sublinear static regret under strongly convex losses. Given this reduction, we see that these algorithms can also be used to achieve sublinear static regret in the converging supervisor setup if the extra term is sublinear. Chapter 5.3.1 identifies when this is the case.

Corollary 5.3.1. *If* $\forall s \in \mathcal{S},\ \forall N > i, \|\psi_i(s) - \psi_N(s)\| \leq f_i$ *where* $\lim_{i\to\infty} f_i = 0$*, then* $\text{Regret}_N^S(\psi_N)$ *can be decomposed as follows:*

$$\text{Regret}_N^S(\psi_N) = \text{Regret}_N^S((\psi_i)_{i=1}^N) + o(N)$$

5.3.2 Dynamic Regret

Although the static regret analysis provides a bound on the average loss, the quality of that bound depends on the term $\min_\theta \sum_{i=1}^N l_i(\pi_\theta, \psi_N)$, which in practice is often very large due to approximation error between the policy class and the actual supervisor. Furthermore, it has been shown that despite sublinear static regret, policy learning may be unstable under certain dynamics [125, 128]. Recent analyses have turned to dynamic regret [124, 125], which measures the sub-optimality of a policy on its own distribution: $l_i(\pi_{\theta_i}, \psi_N) - \min_\theta l_i(\pi_\theta, \psi_N)$. Thus, low dynamic regret shows that a policy is on average performing optimally on its own distribution. This framework also helps determine if policy learning will be stable or if convergence is possible [124]. However, these notions require understanding the sensitivity of the MDP to changes in the policy. We quantify this with an additional Lipschitz assumption on the trajectory distributions induced by the policy as in [124, 125, 129]. We show that even in the converging supervisor setting, it is possible to achieve sublinear dynamic regret given this additional assumption and a converging supervisor by reducing the problem to a predictable online learning problem [129]. Note that this yields the surprising result that it is possible to do well on each round even against a dynamic comparator which has labels from the last observed supervisor. The standard notion of dynamic regret is given in Chapter 5.3.3 below.

Definition 5.3.3. *The dynamic regret with respect to the sequence of supervisors* $(\psi_i)_{i=1}^N$ *is given by the difference in the performance of policy* π_{θ_i} *and that of the best policy under the current round's loss, which compares the performance of current policy* π_{θ_i} *and current supervisor* ψ_i*.*

$$\text{Regret}_N^D((\psi_i)_{i=1}^N) = \sum_{i=1}^N l_i(\pi_{\theta_i}, \psi_i) - \sum_{i=1}^N l_i(\pi_{\theta_i^*}, \psi_i) \ \textit{where}\ \theta_i^* = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_i)$$

However, similar to the static regret analysis in Section 5.3.1, we seek to analyze the dynamic regret with respect to labels from the last observed supervisor ψ_N, which is defined as follows.

Definition 5.3.4. *The dynamic regret with respect to supervisor ψ_N is given by the difference in the performance of policy π_{θ_i} and that of the best policy under the current round's loss, which compares the performance of current policy π_{θ_i} and last observed supervisor ψ_N.*

$$\text{Regret}_N^D(\psi_N) = \sum_{i=1}^{N} l_i(\pi_{\theta_i}, \psi_N) - \sum_{i=1}^{N} l_i(\pi_{\theta_i^\star}, \psi_N) \textit{ where } \theta_i^\star = \arg\min_{\theta \in \Theta} l_i(\pi_\theta, \psi_N)$$

We first show that there is a reduction from $\text{Regret}_N^D(\psi_N)$ to $\text{Regret}_N^D((\psi_i)_{i=1}^N)$.

Lemma 5.3.1. $\text{Regret}_N^D(\psi_N)$ *can be bounded above as follows:*

$$\text{Regret}_N^D(\psi_N) \leq \text{Regret}_N^D((\psi_i)_{i=1}^N) + 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right]$$

Given the notion of supervisor convergence discussed in Chapter 5.3.1, Chapter 5.3.2 shows that if we can achieve sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$, we can also achieve sublinear $\text{Regret}_N^D(\psi_N)$.

Corollary 5.3.2. *If* $\forall s \in \mathcal{S}, \ \forall N > i, \|\psi_i(s) - \psi_N(s)\| \leq f_i$ *where* $\lim_{i\to\infty} f_i = 0$*, then* $\text{Regret}_N^D(\psi_N)$ *can be decomposed as follows:*

$$\text{Regret}_N^D(\psi_N) = \text{Regret}_N^D((\psi_i)_{i=1}^N) + o(N)$$

It is well known that $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ cannot be sublinear in general [124]. However, as in [124, 125], we can obtain conditions for sublinear regret by leveraging the structure in the imitation learning problem with a Lipschitz continuity condition on the trajectory distribution. Let $d_{TV}(p, q) = \frac{1}{2} \int |p - q| d\tau$ denote the total variation distance between two trajectory distributions p and q.

Assumption 5.3.1. *There exists $\eta \geq 0$ such that the following holds on the trajectory distributions induced by policies parameterized by θ_1 and θ_2:*

$$d_{TV}(p(\tau|\theta_1), p(\tau|\theta_2)) \leq \eta \|\theta_1 - \theta_2\|/2$$

A similar assumption is made by popular RL algorithms [133, 134], and Chapter 5.3.2 shows that with it, sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ can be achieved using results from predictable online learning [129].

Lemma 5.3.2. *If Assumption 5.3.1 holds and $\alpha > 4G\eta \sup_{a \in \mathcal{A}} \|a\|$, then there exists an algorithm where* $\text{Regret}_N^D((\psi_i)_{i=1}^N) = o(N)$*. If the diameter of the parameter space is bounded, the greedy algorithm, which plays $\theta_{i+1} = \arg\min_{\theta \in \Theta} l_i(\pi_\theta, \psi_N)$, achieves sublinear* $\text{Regret}_N^D((\psi_i)_{i=1}^N)$*. Furthermore, if the losses are γ-smooth in θ and $\frac{4G\eta \sup_{a \in \mathcal{A}} \|a\|}{\alpha} > \frac{\alpha}{2\gamma}$, then online gradient descent achieves sublinear* $\text{Regret}_N^D((\psi_i)_{i=1}^N)$*.*

Finally, we combine the results of Chapter 5.3.2 and Chapter 5.3.2 to conclude that since we can achieve sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ and have found a reduction from $\text{Regret}_N^D(\psi_N)$ to $\text{Regret}_N^D((\psi_i)_{i=1}^N)$, we can also achieve sublinear dynamic regret in the converging supervisor setting.

Theorem 5.3.2. *If* $\forall s \in \mathcal{S}$, $\forall N > i, \|\psi_i(s) - \psi_N(s)\| \le f_i$ *where* $\lim_{i \to \infty} f_i = 0$ *and under the assumptions in Lemma 5.3.2, there exists an algorithm where* $\text{Regret}_N^D(\psi_N) = o(N)$*. If the diameter of the parameter space is bounded, the greedy algorithm that plays* $\theta_{i+1} = \arg\min_{\theta \in \Theta} l_i(\pi_\theta, \psi_N)$ *achieves sublinear* $\text{Regret}_N^D(\psi_N)$*. Furthermore, if the losses are* γ*-smooth in* θ *and* $\frac{4G\eta \sup_{a \in \mathcal{A}} \|a\|}{\alpha} > \frac{\alpha}{2\gamma}$*, then online gradient descent achieves sublinear* $\text{Regret}_N^D(\psi_N)$*.*

5.4 Converging Supervisors for Deep Continuous Control

Sun et al. [121] apply DPI to continuous control tasks, but assume that both the learner and supervisor are of the same policy class and from a class of distributions for which computing the KL-divergence is computationally tractable. These constraints on supervisor structure limit model capacity compared to state-of-the-art deep RL algorithms. In contrast, we do not constrain the structure of the supervisor, making it possible to use any converging, improving supervisor (algorithmic or human) with no additional engineering effort. Note that while all provided guarantees only require that the supervisor *converges*, we implicitly assume that the supervisor labels actually *improve* with respect to the MDP reward function, R, when trained with data on the learner's distribution for the learner to achieve good task performance. This assumption is validated by the experimental results in this chapter and those in prior work [119, 120]. One strategy to encourage the supervisor to improve on the learner's distribution is to add noise to the learner policy to increase the variety of the experience used by the supervisor to learn information such as system dynamics. However, this was not necessary for the environments considered in this chapter, and we defer further study in this direction to future work.

We utilize the converging supervisor framework (CSF) to motivate an algorithm that uses the state-of-the-art deep MBRL algorithm, PETS, as an improving supervisor. Note that while for analysis we assume a deterministic supervisor, PETS produces stochastic supervision for the agent. We observe that this does not detrimentally impact performance of the policy in practice. PETS was chosen since it has demonstrated superior data efficiency compared to other deep RL algorithms [25]. We collect policy rollouts from a model-free learner policy and refit the policy on each episode using DAgger [122] with supervision from PETS, which maintains a trained dynamics model based on the transitions collected by the learner. Supervision is generated via MPC by using the cross entropy method to plan over the learned dynamics for each state in the learner's rollout, but is collected after the rollout has completed rather than at each timestep of every policy rollout to reduce online computation time.

5.5 Experiments

The method presented in Section 5.4 uses the Converging Supervisor Framework (CSF) to train a learner policy to imitate a PETS supervisor trained on the learner's distribution. We expect the CSF learner to be less data efficient than PETS, but have significantly faster policy evaluation time. To evaluate this hypothesis, we measure the gap in data efficiency between the learner on its own distribution (CSF learner), the supervisor on the learner's distribution (CSF supervisor) and the supervisor on its own distribution (PETS). Returns for the CSF learner and CSF supervisor are computed by rolling out the model-free learner policy and model-based controller after each training episode. Because the CSF supervisor is trained with off-policy data from the learner, the difference between the performance of the CSF learner and CSF supervisor measures how effectively the CSF learner is able to track the CSF supervisor's performance. The difference in performance between the CSF supervisor and PETS measures how important on-policy data is for PETS to generate good labels. All runs are repeated 3 times to control for stochasticity in training; see supplementary material for further experimental details. The DPI algorithm in Sun et al. [121] did not perform well on the presented environments, so we do not report a comparison to it. However, we compare against the following set of 3 state-of-the-art model-free and model-based RL baselines and demonstrate that the CSF learner maintains the data efficiency of PETS while reducing online computation time significantly by only collecting policy rollouts from the fast model-free learner instead of from the PETS supervisor.

1. **Soft Actor Critic (SAC)**: State-of-the-art maximum entropy model-free RL algorithm [62].
2. **Twin Delayed Deep Deterministic policy gradients (TD3)**: State-of-the-art model-free RL algorithm [63] which uses target networks and delayed policy updates to improve DDPG [135], a popular actor critic algorithm.
3. **Model-Ensemble Trust Region Policy Optimization (ME-TRPO)**: State-of-the-art model-free, model-based RL hybrid algorithm using a set of learned dynamics models to update a closed-loop policy offline with model-free RL [134].

5.5.1 Simulation Experiments

We consider the PR2 Reacher and Pusher continuous control MuJoCo domains from Chua et al. [25] (Figure 5.1) since these are standard benchmarks on which PETS attains good performance. For both tasks, the CSF learner outperforms other state-of-the-art deep RL algorithms, demonstrating that the CSF learner enables fast policy evaluation while maintaining data efficient learning. The CSF learner closely matches the performance of both the CSF supervisor and PETS, indicating that the CSF learner has similar data efficiency

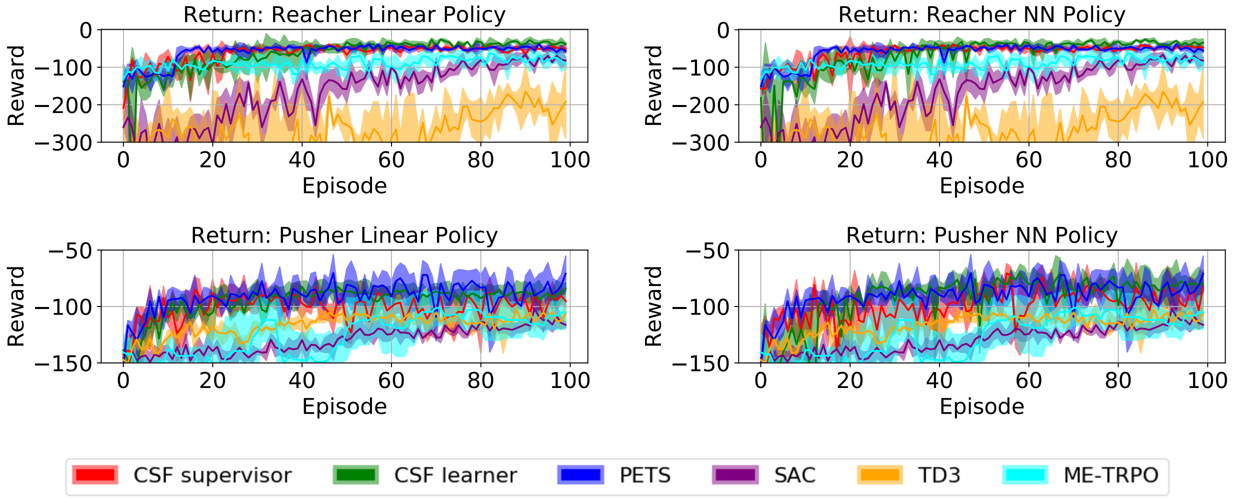


Figure 5.1: **Simulation experiments:** Training curves for the CSF learner, CSF supervisor, PETS, and baselines for the MuJoCo Reacher (top) and Pusher (bottom) tasks for a linear (left) and neural network (NN) policy (right). The linear policy is trained via ridge-regression with regularization parameter $\alpha = 1$, satisfying the strongly-convex loss assumption in Section 5.2. To test more complex policy representations, we repeat experiments with a neural network (NN) learner with 2 hidden layers with 20 hidden units each. The CSF learner successfully tracks the CSF supervisor on both domains, performs well compared to PETS, and outperforms other baselines with both policy representations. The CSF learner is slightly less data efficient than PETS, but policy evaluation is up to 80x faster than PETS. SAC, TD3, and ME-TRPO use a neural network policy/dynamics class.

as PETS. Results using a neural network CSF learner suggest that losses strongly-convex in θ may not be necessary in practice.

This result is promising because if the model-free learner policy is able to achieve similar performance to the supervisor on its own distribution, we can simultaneously achieve the data efficiency benefits of MBRL and the low online computation time of model-free methods. To quantify this speedup, we present timing results in Table 5.1, which demonstrate that a significant speedup (up to 80x in this case) in policy evaluation is possible. Note that although we still need to evaluate the model-based controller on each state visited by the learner to generate labels, since this only needs to be done offline, this can be parallelized to reduce offline computation time as well.

5.5.2 Physical Robot Experiments

We also test CSF with a neural network policy on a physical da Vinci Surgical Robot (dVRK) [78] to evaluate its performance on multi-goal tasks where the end effector must be controlled to desired positions in the workspace. We evaluate the CSF learner/supervisor and PETS on the physical robot for both single and double arm versions of this task, and

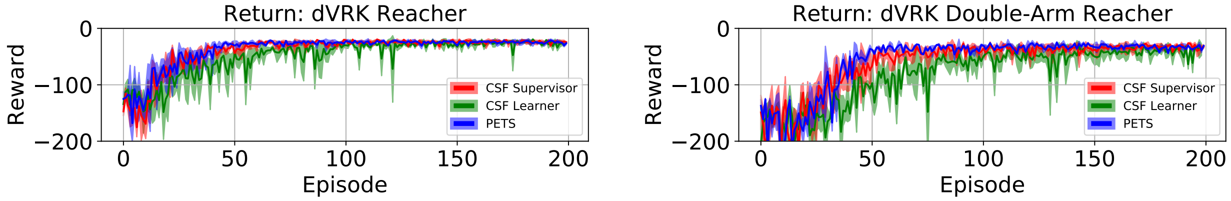


Figure 5.2: **Physical experiments:** Training curves for the CSF learner, CSF supervisor and PETS on the da Vinci surgical robot with a neural network policy. The CSF learner is able to track the CSF supervisor and PETS effectively and can be queried up to 20x faster than PETS. However, due to control frequency limitations on this system, the CSF learner has a policy evaluation time that is only 1.52 and 1.46 times faster than PETS for the single and double-arm tasks respectively. The performance gap between the CSF learner and the supervisor takes longer to diminish for the harder double-arm task.

Table 5.1: **Policy evaluation and query times:** We report policy evaluation times in seconds over 100 episodes for the CSF learner and PETS (format: mean ± standard deviation). Furthermore, for physical experiments, we also report the total time taken to query the learner and PETS over an episode, since this difference in query times indicates the true speedup that CSF can enable (format: (total query time, policy evaluation time)). Policy evaluation and query times are nearly identical for simulation experiments. We see that the CSF learner is 20-80 times faster to query than PETS across all tasks. Results are reported on a desktop running Ubuntu 16.04 with a 3.60 GHz Intel Core i7-6850K and a NVIDIA GeForce GTX 1080. We use the NN policy for all timing results.

	PR2 Reacher (Sim)	PR2 Pusher (Sim)	dVRK Reacher	dVRK Double-Arm Reacher
CSF Learner	$\mathbf{0.29 \pm 0.01}$	$\mathbf{1.13 \pm 0.66}$	$(\mathbf{0.036 \pm 0.009}, \mathbf{5.54 \pm 0.67})$	$(\mathbf{0.038 \pm 0.007}, \mathbf{8.87 \pm 1.12})$
PETS	24.77 ± 0.08	57.77 ± 17.12	$(0.78 \pm 0.02, 8.43 \pm 1.07)$	$(0.88 \pm 0.07, 12.97 \pm 0.77)$

find that the CSF learner is able to track the PETS supervisor effectively (Figure 5.2) and provide up to a 22x speedup in policy query time (Table 5.1). We expect the CSF learner to demonstrate significantly greater speedups relative to standard deep MBRL for higher dimensional tasks and for systems where higher-frequency commands are possible.

5.6 Conclusion

We formally introduce the converging supervisor framework for on-policy imitation learning and show that under standard assumptions, we can achieve sublinear static and dynamic regret against the best policy in hindsight with labels from the last observed supervisor, even when labels are only provided by the converging supervisor during learning. We then show a connection between the converging supervisor framework and DPI, and use this to present an algorithm to accelerate policy evaluation for model-based RL without making any assumptions on the structure of the supervisor. We use the state-of-the-art deep MBRL algorithm, PETS, as an improving supervisor and maintain its data efficiency while significantly accelerating policy evaluation. Finally, we evaluate the efficiency of the method by successfully training a policy on a multi-goal reacher task directly on a physical surgical robot. The provided analysis and framework suggests a number of interesting questions regarding the degree to which non-stationary supervisors affect policy learning. In future work, it would be interesting to derive specific convergence guarantees for the converging supervisor setting, consider different notions of supervisor convergence, and study the trade-offs between supervision quality and quantity.

Part II

Safe Exploration Using Prior Failures

Chapter 6

Learning a Recovery Zone and Policy

In this chapter, we will form a different flavor of safe set that maintains dynamical distance from constraint violating states. When the policy attempts to leave the safe set, a recovery policy is invoked, and the robot is guided back to safety before task-driven exploration can continue. This type of safe set enables much less constrained exploration, and it is also compatible with many different types of reinforcement learning algorithms.

6.1 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 [52], locomotion [89], and manipulation [26, 91]. However, when deploying RL agents in the real world, unconstrained exploration can result in highly suboptimal behaviors which can damage the robot, break surroundings objects, or bottleneck the learning process. For example, consider an agent tasked with learning to extract a carton of milk from a fridge. If it tips over the carton, then not only can this possibly break the carton and create a mess, but it also requires laborious human effort to wipe up the milk and replace the carton so that the robot can continue learning. In the meantime, the robot is not able to collect experience or improve its policy until the consequences of this violation are rectified. Thus, endowing RL agents with the ability to satisfy constraints during learning not only enables robots to interact safely, but also allows them to more efficiently learn in the real world. However, enforcing constraints on the agent's behavior during learning is challenging, since system dynamics and the states leading to constraint violations may be initially unknown and must be learned from experience, especially when learning from high dimensional observations such as images. Safe exploration poses 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.

We consider a RL formulation subject to constraints on the probability of unsafe future behavior and design an algorithm that can balance the often conflicting objectives of task-

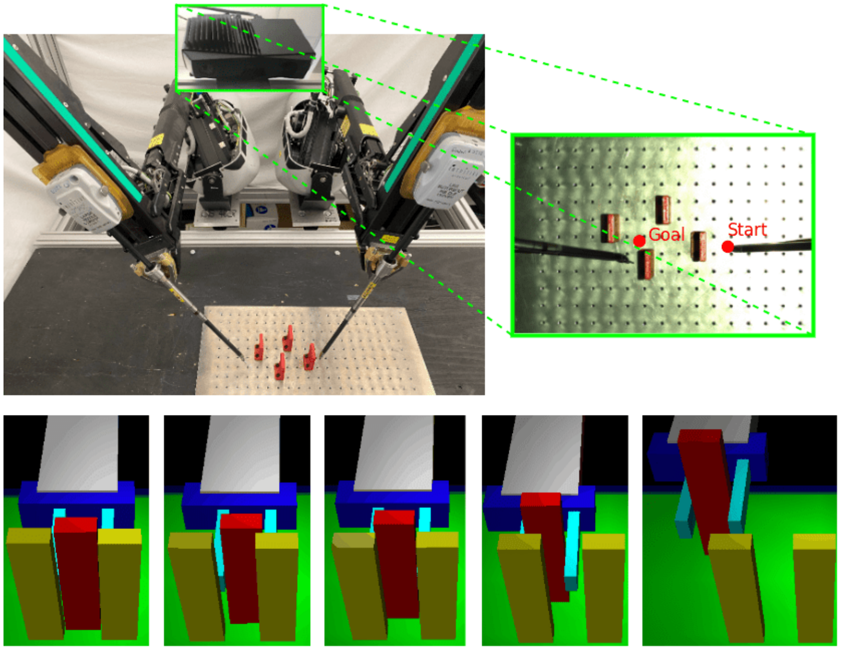


Figure 6.1: Recovery RL can safely learn policies for contact-rich tasks from high-dimensional image observations in simulation experiments and on a physical robotic system. We evaluate Recovery RL on an image-based obstacle avoidance task with delta-position control on the da Vinci Research Kit (top left) with overhead image observations (top right). We find that Recovery RL substantially outperforms prior methods (Figure 6.6), suggesting that it can be used for visuomotor control on physical robots. We also find that Recovery RL can perform challenging contact-rich manipulation tasks in simulation; as shown in the bottom row, Recovery RL successfully extracts the red block without toppling other blocks by learning to nudge it away from other blocks before grasping it.

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 use in practice. First, the inherent objective conflict between exploring to learn new tasks and limiting exploration to avoid constraint violations can lead to suboptimalities in policy optimization. Second, exploring the environment to learn about constraints requires a significant amount of constraint violations during learning. However, this can result in the agent taking uncontrolled actions which can damage 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 [11, 12, 136, 137], 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 [138], the recovery policy can be viewed as defining an alternate MDP for the task policy to explore in which constraint violations are unlikely. Separating the task and recovery policies makes it easier to balance task performance and safety, and allows using off-the-shelf RL algorithms for both. 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 under human supervision to illustrate examples of desired behaviors before the agent interacts with the environment or can contain unsafe behaviors previously experienced by the robot in the environment when performing other tasks. Both the recovery set and policy are updated online with agent experience, but the offline data allows the agent to observe constraint violations and learn from them without the task policy directly having to experience too many uncontrolled violations during learning.

We present Recovery RL, a new algorithm for safe robotic RL. Unlike prior work, Recovery RL (1) can 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 - 20 times more efficiently than the next best prior method. We evaluate Recovery RL on an image-based obstacle avoidance task on a physical robot and find that it trades off constraint violations and task successes 3 times more efficiently than the next best prior algorithm.

6.2 Related Work

Prior work has studied safety in RL in several ways, including imposing constraints on expected return [3, 5], risk measures [139–142], and avoiding regions of the MDP where constraint violations are likely [9, 12, 24, 75, 143, 144]. 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 Safety: 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 [5], optimizing a Lagrangian relaxation [3, 6, 145], or constructing Lyapunov functions [146, 147]. The most similar of these works to Recovery RL is Srinivasan et al. [145], which trains a safety critic to estimate 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. [145], 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

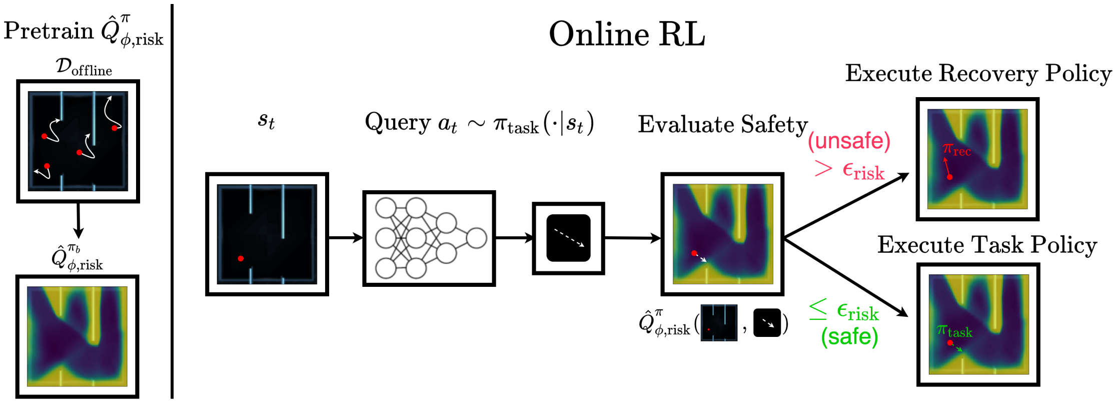

Pretrain $\hat{Q}^{\pi}_{\phi,\text{risk}}$
$\mathcal{D}_{\text{offline}}$
$\hat{Q}^{\pi_b}_{\phi,\text{risk}}$
Online RL
s_t
Query $a_t \sim \pi_{\text{task}}(\cdot|s_t)$
Evaluate Safety
$\hat{Q}^{\pi}_{\phi,\text{risk}}($, $)$
(unsafe)
$> \epsilon_{\text{risk}}$
Execute Recovery Policy
π_{rec}
$\leq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

Figure 6.2: **Recovery RL:** For intuition, 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.

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 [11, 12], which utilize Hamilton-Jacobi reachability analysis to define a task policy and safety controller, and in the context of shielding [136, 137, 148]. In contrast to these works, which assume approximate knowledge of system dynamics or require precise knowledge of constraints apriori, Recovery RL learns information about the MDP, such as constraints and dynamics, from a combination of offline data and online experience. This allows Recovery RL to scale to high-dimensional state spaces such as images, in which exact specification of system dynamics and constraints can be very challenging, and is often impossible. Additionally, Recovery RL reasons about chance constraints rather than robust constraints, which may be challenging to satisfy when dynamics are unknown. Fisac et al. [11] design and prove safety guarantees for learning-based controllers in a robust optimal control setting with known dynamics and a robust control invariant safe set. With these additional assumptions, Recovery RL has similar theoretical properties as well. Han et al. [149] and Eysenbach et al. [9] 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 [9]. However, enforcing the ability to fully reset can be impractical or inefficient. Inspired by this chapter, Recovery RL instead executes approximate resets to nearby safe states when constraint violation is probable. Richter et al. [52] 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: There has also been significant prior work investigating how demonstrations can be leveraged to enable safe exploration. Thananjeyan et al. [14] and Rosolia et al. [15] introduce model predictive control algorithms which leverage initial constraint satisfying demonstrations to iteratively improve their performance with safety guarantees and Thananjeyan et al. [24] 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. Also, Recovery RL is compatible with model-free RL algorithms while [14, 24] require a dynamics model to evaluate reachability-based safety online.

6.3 Problem Statement

We consider RL under Markov decision processes (MDPs), which can be described by 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. 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 extra constraint cost function $C : \mathcal{S} \to \{0, 1\}$ which indicates whether a state is constraint violating and 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 violations, 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 goal is to solve the following constrained optimization problem:

$$\pi^* = \arg\max_{\pi \in \Pi} \{R^\pi : Q^\pi_{\text{risk}}(s_0, a_0) \leq \epsilon_{\text{risk}}\} \tag{6.1}$$

This setting exactly corresponds to the CMDP formulation from [150], 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, $\{\pi : Q^\pi_{\text{risk}} \leq \epsilon\}$, 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 (6.1) 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 assume access to a set of transitions from offline data ($\mathcal{D}_{\text{offline}}$) with examples of constraint violations. Unlike in typical imitation learning settings, this data need not illustrate task successes, but 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.

6.4 Recovery RL

We outline the central ideas behind Recovery RL. In Section 6.4.1, we review how to learn a safety critic to estimate the probability of future constraint violations for the agent's policy. Then in Section 6.4.2, we show how this safety critic is used to define the recovery policy for Recovery RL and the recovery set in which it is activated. In Section 6.4.3 we discuss how the safety critic and recovery policy are initialized from offline data and in Section 6.4.4 we discuss implementation details. See Algorithm 3 and Figure 6.2 for further illustration of Recovery RL.

6.4.1 Preliminaries: Training a Safety Critic

As in Srinivasan et al. [145], Recovery RL learns a critic function Q^{π}_{risk} that estimates the discounted future probability of constraint violation of the current policy π:

$$\begin{aligned} Q^{\pi}_{\text{risk}}(s_t, a_t) &= \mathbb{E}_{\pi}\left[\sum_{t'=t}^{\infty} \gamma_{\text{risk}}^{t'-t} c_{t'} | s_t, a_t\right] \\ &= c_t + (1 - c_t)\gamma_{\text{risk}} \mathbb{E}_{\pi}\left[Q^{\pi}_{\text{risk}}(s_{t+1}, a_{t+1}) | s_t, a_t\right]. \end{aligned} \tag{6.2}$$

Here $c_t = 1$ indicates that state s_t is constraint violating with $c_t = 0$ otherwise. Note we do not assume access to the true constraint cost function C. This is different from the standard Bellman equations to the assumption that episodes terminate when $c_t = 1$. 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_t).

We train $\hat{Q}^{\pi}_{\phi,\text{risk}}$ by minimizing the following MSE loss with respect to the target (RHS of equation 6.2).

$$\begin{aligned} J_{\text{risk}}(s_t, a_t, s_{t+1}; \phi) = \frac{1}{2}\Big(&\hat{Q}^{\pi}_{\phi,\text{risk}}(s_t, a_t) - (c_t \\ &+ (1 - c_t)\gamma_{\text{risk}} \mathop{\mathbb{E}}_{a_{t+1} \sim \pi(\cdot | s_{t+1})}[\hat{Q}^{\pi}_{\phi,\text{risk}}(s_{t+1}, a_{t+1})])\Big)^2 \end{aligned} \tag{6.3}$$

and use a target network to create the target values [62, 145].

6.4.2 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 $\mathcal{T}^{\pi}_{\text{rec}}$:

$$\begin{aligned}\mathcal{T}^{\pi}_{\text{safe}} &= \{(s,a) \in \mathcal{S} \times \mathcal{A} : Q^{\pi}_{\text{risk}}(s,a) \le \epsilon_{\text{risk}}\} \\ \mathcal{T}^{\pi}_{\text{rec}} &= \mathcal{S} \times \mathcal{A} \setminus \mathcal{T}^{\pi}_{\text{safe}}\end{aligned}$$

We consider state-action tuple (s,a) to be 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:

$$a_t = \begin{cases} a_t^{\pi_{\text{task}}} & (s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{safe}} \\ a_t^{\pi_{\text{rec}}} & (s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{6.4}$$

Recovery RL filters proposed actions that are likely to lead to unsafe states, equivalent to modifying the environment that π_{task} operates in with new dynamics:

$$P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a) = \begin{cases} P(s'|s,a) & (s,a) \in \mathcal{T}^{\pi}_{\text{safe}} \\ P(s'|s,a^{\pi_{\text{rec}}}) & (s,a) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{6.5}$$

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.

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 6.5? 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.

Algorithm 3 Recovery RL

```
Require: $\mathcal{D}_{\text{offline}}$, task horizon $H$, number of episodes $N$
Pretrain $\pi_{\text{rec}}$ and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on $\mathcal{D}_{\text{offline}}$          ▷ Section 6.4.3
$\mathcal{D}_{\text{task}} \leftarrow \emptyset$, $\mathcal{D}_{\text{rec}} \leftarrow \mathcal{D}_{\text{offline}}$
$s_0 \leftarrow$ env.reset()
for $i \in \{1, \ldots N\}$ do
    for $t \in \{1, \ldots H\}$ do
        if $c_t = 1$ or is_terminal$(s_t)$ then
            $s_t \leftarrow$ env.reset()
        end if
        $a_t^{\pi_{\text{task}}} \sim \pi_{\text{task}}(\cdot|s_t)$          ▷ Query task policy
                                                  ▷ Check if task policy will be unsafe
        if $(s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{rec}}$ then
            $a_t \sim \pi_{\text{rec}}(\cdot|s_t)$          ▷ Select recovery policy
        else
            $a_t = a_t^{\pi_{\text{task}}}$          ▷ Select task policy
        end if
        Execute $a_t$
        Observe $s_{t+1}$, $r_t = R(s_t, a_t)$, $c_t = C(s_t)$
                                                  ▷ Relabel transition
        $\mathcal{D}_{\text{task}} \leftarrow \mathcal{D}_{\text{task}} \cup \{(s_t, a_t^{\pi_{\text{task}}}, s_{t+1}, r_t)\}$
        $\mathcal{D}_{\text{rec}} \leftarrow \mathcal{D}_{\text{rec}} \cup \{(s_t, a_t, s_{t+1}, c_t)\}$
        Train $\pi_{\text{task}}$ on $\mathcal{D}_{\text{task}}$, $\pi_{\text{rec}}$ on $\mathcal{D}_{\text{rec}}$
        Train $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on $\mathcal{D}_{\text{rec}}$          ▷ Eq. 6.3
    end for
end for
```

6.4.3 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, this data can be collected by human defined policies or under human supervision, and thus provide the robotic agent with examples of constraint violations without the robot having to experience too many uncontrolled examples online. We pretrain $\hat{Q}^{\pi}_{\phi,\text{risk}}$ by minimizing Equation 6.3 over offline batches sampled from $\mathcal{D}_{\text{offline}}$. We also pretrain π_{rec} using $\mathcal{D}_{\text{offline}}$. Then, π_{task}, π_{rec}, and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ are all updated online using experience from the agent's composite policy as discussed in Section 6.4.2 and illustrated in Algorithm 3. 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 use a separate set of task demos to initialize π_{task} to expedite learning.

6.4.4 Practical Implementation

Recovery Policy: Any off-policy RL algorithm can be used to learn π_{rec}. In this chapter, 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 [135]. For model-based recovery, we perform model predictive control (MPC) over a learned dynamics model f_θ using the safety critic as a cost function. For lower dimensional tasks, we utilize the PETS algorithm from Chua et al. [25] to plan over a learned stochastic dynamics model, while for tasks with visual observations, we use a VAE based latent dynamics model. **Task Policy:** We utilize the popular maximum entropy RL algorithm SAC [62] to learn π_{task}, but note that any RL algorithm could be used. Details on the implementation of both policies is in the supplement.

6.5 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 both and (2) effectively use offline data for safe RL.

Domains: We evaluate Recovery RL on a set of 6 simulation domains (Figure 6.3) and an image-based obstacle avoidance task on a physical robot (Figure 6.6). 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 obstacle avoidance task on the da Vinci Research Kit (dVRK) [78] where the robot must guide its end effector within 2 mm of a target position from two possible starting locations without touching red 3D printed obstacles in the workspace. See Figure 6.1 for an illustration of the experimental setup. The dVRK is cable-driven and has relatively imprecise controls, motivating closed-loop control strategies to compensate for these errors [151]. Furthermore, the dVRK system has been used in the past to evaluate safe RL algorithms [24] due to its high cost and the delicate structure of its arms, which make safe learning critical. Further environment, task, and data collection details can be found in the supplement for all simulation and physical experiments.

Offline Data Collection: To effectively initialize $\hat{Q}^{\pi}_{\phi,\text{risk}}$, $\mathcal{D}_{\text{offline}}$ should ideally contain a diverse set of trajectories which violate constraints in different ways. Since $\mathcal{D}_{\text{offline}}$ need not be task specific, data from other tasks in the environment could be used, or simple human defined policies can be used to illustrate constraint violating behaviors. We take the latter approach: for all navigation environments (Navigation 1, Navigation 2, Maze, Image Maze, and the physical experiment), offline data is collected by initializing the agent in various regions of the environment and directing the agent towards the closest obstacle. For the object extraction environments (Object Extraction, Object Extraction (Dynamic Obstacle)), demonstrations are collected by guiding the end effector towards the target red block and adding Gaussian noise to controls when it is sufficiently close to the target object to make toppling likely. Recovery RL and all comparisons which have a safety critic are given the same offline dataset $\mathcal{D}_{\text{offline}}$. See the supplementary material for details on the data collection procedure, and the number of total transitions and constraint violating states for all offline datasets.

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. Each run for simulation experiments is replicated across 10 random seeds and we report the mean and standard error. For physical experiments we run each algorithm across 3 random seeds and visualize all 3 runs. In the supplementary material, we also report additional metrics for each experiment: cumulative task successes, cumulative constraint violations, and reward learning curves. We find that Recovery RL violates constraints less often than comparisons while maintaining a similar task success rate and more efficiently optimizing the task reward.

Comparisons: We compare Recovery RL to the following algorithms that ignore constraints (Unconstrained) or enforce constraints via the optimization objective (LR, SQRL,

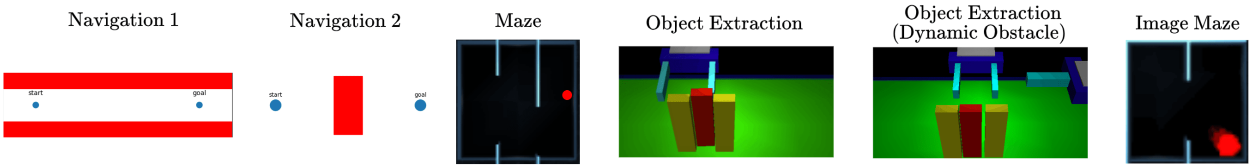


Figure 6.3: **Simulation Experiments Domains:** We evaluate Recovery RL on a set of 2D navigation tasks, two contact rich manipulation environments, and a visual navigation task. In Navigation 1 and 2, the goal is to navigate from the start set to the goal set without colliding into the obstacles (red) 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).

RSPO) or via reward shaping (RP, RCPO).

- **Unconstrained**: optimizes task reward, ignoring constraints.
- **Lagrangian Relaxation (LR)**: 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 and the second term approximately enforces $\hat{Q}^{\pi}_{\phi,\text{risk}}(s,a)\leq\epsilon_{\text{risk}}$. Policy parameters and λ are updated via dual gradient descent.
- **Safety Q-Functions for RL (SQRL)** [145]: 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)** [140]: 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 which decreases to 0.
- **Reward Penalty (RP)**: observes a reward function that penalizes constraint violations: $R'(s,a)=R(s,a)-\lambda C(s)$.
- **Critic Penalty Reward Constrained Policy Optimization (RCPO)** [3]: 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 [62]) 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. Details on Recovery RL and all comparison algorithms are in the supplement.

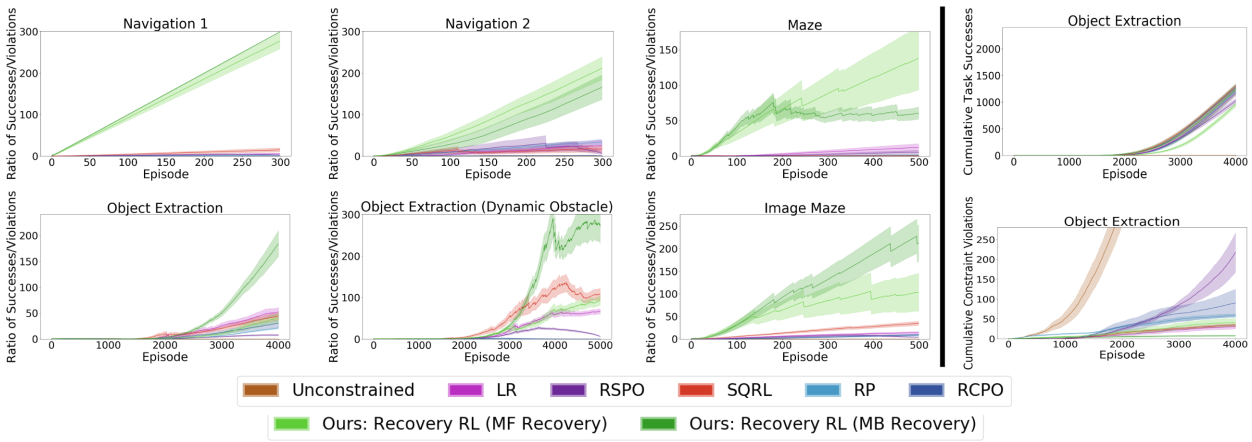


Figure 6.4: **Simulation Experiments: Left: ratio of successes to constraint violations over the course of online training.** In all navigation tasks, we find that Recovery RL significantly outperforms prior methods with both model-free and model-based recovery policies, while for the object extraction 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 $\hat{Q}^{\pi}_{\phi,\text{risk}}$. Results are averaged over 10 runs for each algorithm; the sawtooth pattern occurs due to constraint violations, which result in a sudden drop in the ratio. **Right: cumulative successes and constraint violations.** Additionally, we show the cumulative task successes and cumulative constraint violations for the Object Extraction task for all algorithms, and find that Recovery RL with model-based recovery succeeds more often than all comparisons while also violating constraints the least. Similar plots for all other experimental domains can be found in the supplementary material.

Results: We first study the performance of Recovery RL and prior methods in all simulation domains in Figure 6.4. 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 mechanism is better able to compensate for approximation errors in $\hat{Q}^{\pi}_{\phi,\text{risk}}$, resulting in a more robust recovery policy. We find that the prior methods often get very low ratios since they tend to achieve a similar number of task completions as Recovery RL, but with many more constraint violations. In contrast, Recovery RL is generally able to effectively trade off between task performance and safety. This is illustrated on the right pane of Figure 6.4, which suggests that Recovery RL with model-based recovery not only succeeds more often than comparison algorithms, but also exhibits fewer constraint violations. We study this further in the supplement. Finally, we evaluate Recovery RL and prior algorithms on the image-based obstacle avoidance task

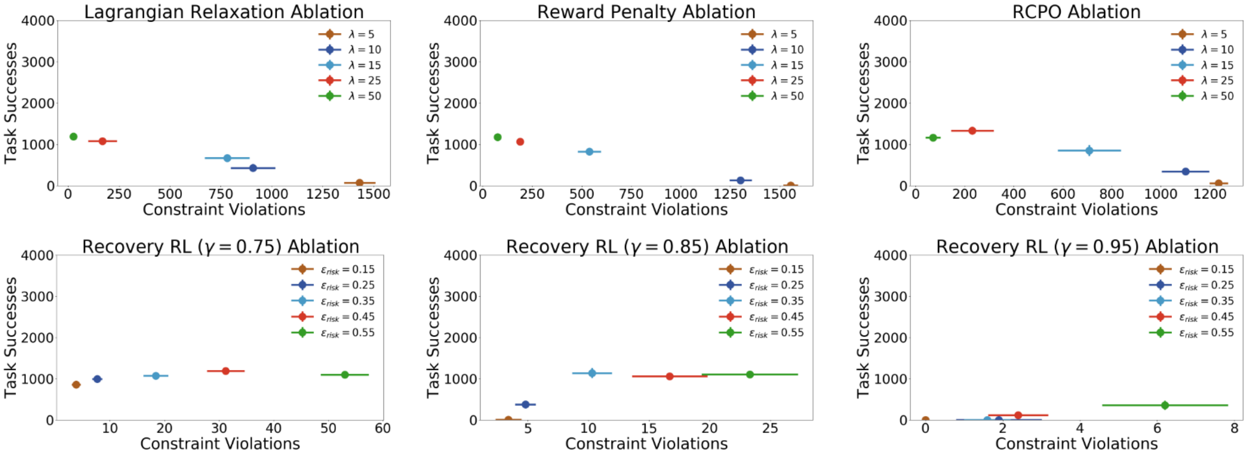


Figure 6.5: **Sensitivity Experiments:** We report the final number of task successes and constraint violations averaged over 10 runs at the end of training for Recovery RL and comparison algorithms for a variety of different hyperparameter settings on the Object Extraction task. We find that the comparison 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.

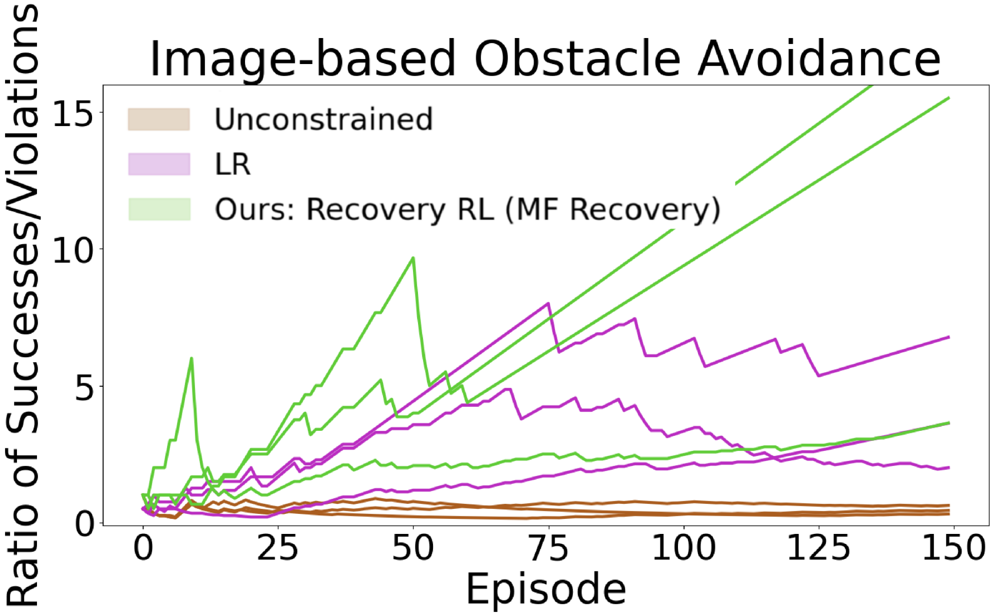


Figure 6.6: **Physical Experiment**: We evaluate Recovery RL on an image-based obstacle avoidance task (red obstacles) on the dVRK (Figure 6.1). We supply all algorithms with an overhead RGB image as input and run each algorithm 3 times. We find that Recovery RL significantly outperforms Unconstrained and LR.

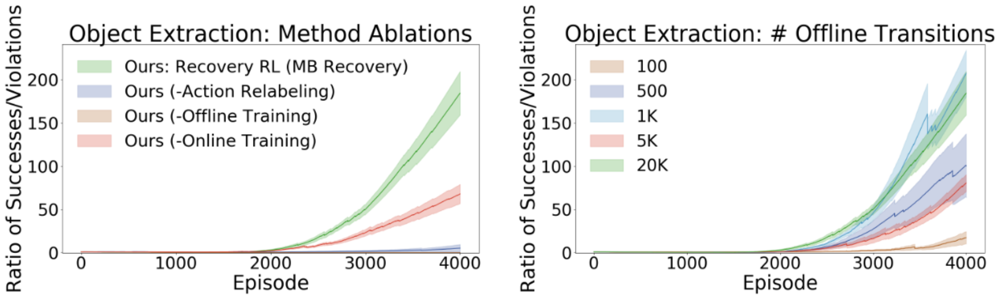


Figure 6.7: **Ablations:** We first study the affect of different algorithmic components of Recovery RL (left). Results suggest that offline pretraining of π_{rec} and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ is critical for good performance, while removing online updates leads to a much smaller reduction in performance. Furthermore, we find that the action relabeling method for training π_{task} (Section 6.4.2) is critical for good performance. We then study the sensitivity of Recovery RL 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.

illustrated in Figure 6.1 and find that Recovery RL substantially outperforms prior methods, suggesting that Recovery RL can be used for contact-rich visuomotor control tasks in the real world (Figure 6.6). We study when Recovery RL violates constraints in the supplement, and find that in most tasks, the recovery policy is already activated when constraint violations occur. This is encouraging, because if a recovery policy is challenging to learn, Recovery RL could still be used to query a human supervisor for interventions.

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 6.7. 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 6.4.2). 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.

Sensitivity Experiments: We tune hyperparameters for Recovery RL and all baselines to ensure a fair comparison. We first tune γ_{risk} and ϵ_{risk} for Recovery RL, and then use the same γ_{risk} and ϵ_{risk} for prior methods to ensure that all algorithms use the same safety critic training procedure. These two hyperparameters are the only ones tuned for Recovery RL and SQRL. For RP, RCPO, and LR, 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 comparison to 0 with an evenly spaced linear schedule over all training episodes. In Figure 6.5, 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. Recovery RL appears less sensitive to hyperparameters than the comparisons for the γ_{risk} values we consider.

6.6 Conclusion

We present Recovery RL, a new algorithm for safe RL which is able to more effectively balance task performance and constraint satisfaction than 5 state-of-the-art prior algorithms for safe RL across 6 simulation domains and an image-based obstacle avoidance task on a physical robot. In future work we hope to explore further evaluation on physical robots, establish formal guarantees, and use ideas from offline RL to more effectively pretrain the recovery policy. We will explore settings in which constraint violations may not be catastrophic and applications for large-scale robot learning.

Chapter 7

Transferring a Recovery Zone and Policy from Other Tasks

In this chapter, we will discuss how to transfer the safety critic and recovery policy learned from similar dynamical systems to new ones without requiring as much data, specifically of constraint violations, to do so. The proposed algorithm uses recent techniques from meta learning to rapidly adapt the safety critic to test environments that have similar but different dynamics.

7.1 Introduction

Reinforcement learning (RL) is a versatile abstraction that has shown significant recent success in learning a variety of different robotic tasks purely from interactions with the environment. However, while learning policies through online experience affords simplicity and generality to RL algorithms, this can result in unsafe behavior during online learning. Unconstrained exploration can potentially lead to highly unproductive or unsafe behaviors, which can cause equipment/monetary losses, risk to surrounding humans, and inhibit the learning process. This motivates safe RL algorithms that leverage prior experience to avoid unsafe behaviors during exploration. Recent work on safe RL algorithms typically learn a risk measure [4, 8, 145], which captures the probability that an agent will violate a constraint in the future, and then uses this measure to avoid unsafe behaviors. For example, a robot may realize that, under its current policy, it is likely to collide with a wall and hence take preemptive measures to avoid collision. However, the agent's ability to be safe largely depends on the accuracy of the learned risk measure, and learning this risk measure requires significant data demonstrating unsafe behavior. This poses a key challenge: to know how to be safe, an agent must see sufficiently many examples of unsafe behavior, but the more such examples it generates, the less effectively it has protected itself from unsafe behaviors.

This challenge motivates developing methods to endow RL agents with knowledge about constraints *before* online interaction, so the agent can learn safely without excessive constraint violations during deployment in risk-sensitive environments. Prior work studies how to use previous data of agent interactions, either via online interaction or offline datasets, to learn a risk measure which can then be adapted during online deployment [8, 108, 145]. However, a challenge with these methods is that these offline transitions are required to be in the same environment as that in which the agent is deployed, which may not always be practical in risk-sensitive environments in which a large number of constraint violations could be exceedingly costly or dangerous. Additionally, shifting dynamics is a ubiquitous phenomenon in real robot hardware: for example losses in battery voltage [152] or wear-and-tear in manipulators or actuators [153]. These changes can drastically change the space of safe behaviors, as the robot may need to compensate for unforeseen differences in the robot dynamics. Furthermore, these changes in dynamics will often not be immediately observable for a robot control policy, motivating algorithms which can identify and adapt to these changes based online interaction.

To address this, we aim to effectively transfer knowledge about safety between environments with different dynamics, so that when learning some downstream task in a test environment with previously unseen dynamics, the agent can rapidly learn to be safe. Our insight is that the agent should be able to leverage offline datasets across previous deployments, with knowledge of only the safety of states in these datasets, to rapidly learn to be safe in new environments without task specific information. The contributions of this chapter are (1) casting safe RL as an offline meta-reinforcement learning problem [154, 155], where the objective is to leverage fully offline data from training and test environments to learn how to be safe in the test environment; (2) MEta-learning for Safe Adaptation (MESA), which meta-learns a risk measure that is used for safe reinforcement learning in new environments with previously unseen dynamics; (3) simulation experiments across 3 continuous control domains which suggest that MESA can cut the number of constraint violations in half in a new environment with previously unseen dynamics while maintaining task performance compared to prior algorithms. Please see the supplement for a more thorough discussion of related work.

7.2 Related Work

7.2.1 Safe Reinforcement Learning

There has been significant recent work on reinforcement learning algorithms which can satisfy safety constraints. We specifically focus on satisfying explicit state-space constraints in the environment and review prior literature which also considers this setting [2]. Prior work has considered a number of methods for incorporating constraints into policy optimization for reinforcement learning, including trust region based methods [5, 156], optimizing a Lagrangian relaxation [3, 4, 6, 145], drawing connections to Lyapunov theory [75, 146, 147], anticipating violations with learned dynamics models [14, 24, 108, 157], using Gaussian processes to reason about uncertainty [158, 159], using recovery policies to shield the agent from constraint violations [8, 9, 136, 137, 149], formal reachability analysis [11, 12, 160–163], or formal logic [148, 164]. Zhang et al. [108] design a model-based RL algorithm which leverages unsafe data from a variety of training environments with different dynamics to predict whether the agent will encounter unsafe states and penalize its reward if this is the case. Unlike Zhang et al. [108], we explicitly optimize for adaptation and decouple information about constraints from the reward function, making it possible to efficiently learn transferable notions of safety. Additionally, we learn a risk measure in a fully offline setting, and do not assume direct access to the training environments.

Srinivasan et al. [145] introduce the idea of a safety critic, which estimates the discounted probability of constraint violation of the current policy given the current state and a proposed action. Bharadhwaj et al. [4], Thananjeyan et al. [8], and Srinivasan et al. [145] present 3 different methods to utilize the learned safety critic for safe RL. Thananjeyan et al. [8] and Srinivasan et al. [145] also leverage prior data from previous interactions to learn how to be safe. However, unlike these works, which assume that prior data is collected in an environment with the same dynamics as the test environment, MESA learns to leverage experience from a variety of environments with different dynamics in addition to a small amount of data from the test environment. This choice makes it possible to avoid excessive constraint violations in the test environment, in which constraint violations may be costly, by leveraging prior experience in safer environments or from accident logs from previous deployments.

7.2.2 Meta Reinforcement Learning

There is a rich literature [165–169] studying learning agents that can efficiently adapt to new tasks. In the context of reinforcement learning, this problem, termed *meta-reinforcement learning* [170–172], aims to learn RL agents which can efficiently adapt their policies to new environments with unseen transition dynamics and rewards. A number of strategies exist to accomplish this such as recurrent or recursive policies [170, 171, 173], gradient based optimization of policy parameters [172, 174], task inference [175–177], or adapting dynamics models for model-based RL [26, 178]. One of the core challenges studied in many meta-RL works is efficient exploration [175, 179–181], since the agent needs to efficiently explore its new environment to identify the underlying task. Unlike all of these prior works, which focus on learning transferable policies, we focus on learning *risk measures* which can be used to safely learn new tasks in a test environment with previously unseen dynamics. Additionally, we study learning these measures in the context of offline meta-RL, and learn from purely offline datasets of prior interactions in various environments with different dynamics.

The *offline meta reinforcement learning* problem [154, 155, 182] considers a setting in which the agent learns from a set of offline data from each training task, and adapts to the test environment conditioned only on a small set of offline transitions. Critically, this setting is particularly well suited to the problem of safe RL, because it has potential to enable an agent to be safe in an environment with previously unseen dynamics conditioned on a small set of experiences from that environment. In this chapter, we formalize safe reinforcement learning as an offline meta-RL problem and present an algorithm to adapt *a safety critic* to new environments and use this adapted safety critic for safe reinforcement learning. One option for meta-learning for safe RL is using meta-learning for sim-to-real domain adaptation where data can be collected safely and at scale in simulated environments [183]. By contrast, MESA explicitly reasons about safety constraints in the environment to learn adaptable risk measures. Additionally, while prior work has also explored using meta-learning in the context of safe-RL [184], specifically by learning a single safety filter which keeps policies adapted for different tasks safe, we instead adapt *the risk measure itself* to unseen dynamics and fault structures.

7.3 Preliminaries

7.3.1 Constrained Markov Decision Processes

In safe reinforcement learning, an agent interacts with a Constrained Markov Decision Process (CMDP) [150], defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, C, \rho_0, \gamma, \gamma_{\text{risk}})$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space, the transition dynamics function $P : \mathcal{S}\times\mathcal{A}\times\mathcal{S} \to [0, 1]$ maps the current state and action to a probability distribution of next states, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $C : \mathcal{S} \to \{0, 1\}$ is a constraint function which indicates whether a state is constraint violating, $\rho_0 : \mathcal{S} \to [0, 1]$ is the starting state distribution, and $\gamma, \gamma_{\text{risk}} \in [0, 1]$ are the discount factors for the rewards and constraint values. As in prior work [8, 145], we assume constraint violations end the episode immediately. The expected return for a policy $\pi : \mathcal{S} \to \mathcal{A}$ is $R(\pi) = \mathbb{E}_{\pi,\rho_0,P}\left[\sum_t^\infty \gamma^t r(s_t, a_t)\right]$. The discounted probability of future constraint violation for policy π is $Q^\pi_{\text{risk}}(s_t, a_t) = \mathbb{E}_{\pi,\rho_0,P}\left[\sum_t^\infty \gamma^t_{\text{risk}} C(s_t)\right] = \mathbb{E}_{\pi,\rho_0,P}\left[\sum_t^\infty \gamma^t_{\text{risk}} \mathbb{P}\left(C\left(s_t\right) = 1\right)\right]$. Unlike unconstrained RL, safe RL agents seek to optimize:

$$\pi^* = \arg\max_{\pi} \left\{R^\pi : Q^\pi_{\text{risk}} \le \epsilon_{\text{risk}}\right\} \tag{7.1}$$

where ϵ_{risk} is a hyper-parameter that defines how safe the agent should be.

7.3.2 Safety Critics for Safe RL

Recent work investigates ways to estimate the discounted future probability of catastrophic constraint violation under the current policy: $Q^\pi_{\text{risk}}(s_t, a_t) = \sum_{t'=t}^\infty \gamma^{t'-t}_{\text{risk}} C(s_t)$ [8, 145]. In practice, algorithms search over a parametric function class: $\left\{Q^\pi_{\psi,\text{risk}}(s_t, a_t) : \psi \in \Psi\right\}$, where ψ is a particular parameter vector and Ψ is its possible values. This function is trained by minimizing an MSE loss function with respect to a target function on a dataset of transitions $\{(s_t, a_t, c_t, s_{t+1})_i\}_{i=1}^N$ collected in the environment:

$$\begin{aligned}\mathcal{L}_{\text{risk}}(s_t, a_t, c_t, s_{t+1}) = (&Q^\pi_{\psi,\text{risk}}(s_t, a_t) - (c_t \\ &+ \gamma_{\text{risk}}(1 - c_t)\mathbb{E}_{a_{t+1}\sim\pi(\cdot|s_{t+1})}\left[Q^\pi_{\psi,\text{risk,targ}}(s_{t+1}, a_{t+1})\right]))^2_2\end{aligned}$$

where $Q^{\pi}_{\psi,\text{risk},\text{targ}}$ is a target network and c_t denotes that state s_t is constraint violating. The safety critic can be used for constrained policy search, by either optimizing a Lagrangian function [3, 4, 145] with it or filtering dangerous actions [8, 145].

7.3.3 Recovery RL

In this chapter, we use the safety critic Q^{π}_{risk} to detect when to switch to a recovery policy and to train the recovery policy as in Recovery RL [8]. In particular, Recovery RL trains a task policy π_{task} and a recovery policy π_{rec} and executes actions from π_{task} when the risk estimate is sufficiently low and from π_{rec} otherwise. That is,

$$a_t \sim \begin{cases} \pi_{\text{task}}(\cdot|s_t) & Q^{\pi}_{\text{risk}}(s_t, a^{\pi}_t) \leq \epsilon_{\text{risk}} \\ \pi_{\text{rec}}\ (\cdot|s_t) & \text{otherwise} \end{cases}$$

Here $\epsilon_{\text{risk}} \in [0, 1]$ is a user-specified hyperparameter that indicates the level of risk the agent is willing to take. If the safety critic indicates that the current state and action visited by the task policy is unsafe, the recovery policy will overwrite the task policy's actions, moving the agent back to safe regions of the state space. Both policies can be trained using any reinforcement learning algorithm, where π_{task} optimizes task reward and π_{rec} minimizes Q^{π}_{risk}.

7.3.4 Meta-learning

Consider a task distribution $p(\mathcal{M})$ where tasks are sampled via $\mathcal{M}_i \sim p(\mathcal{M})$. In the RL setting, each task corresponds to an MDP, all of which share the same state and action spaces but may have varying dynamics (e.g. varying controller impedance for a legged robot). The goal in this chapter is to learn risk measures that rapidly adapt to new environments, such as when a robot's actuator loses power and it is forced to compensate with only the remaining actuators. We will briefly discuss how functions can be initialized for rapid adaptation to new tasks by training on similar tasks.

Meta-learning learns a model explicitly optimized for adaptation to a new task from $p(\mathcal{M})$. Let $\theta'_i = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{M}_i}(f_\theta)$ be the parameters θ after a single gradient step from optimizing $\mathcal{L}_{\mathcal{M}_i}(f_\theta)$. Model-Agnostic Meta-Learning (MAML) [172] optimizes the following objective at meta-train time:

$$\min_\theta \mathbb{E}_{\mathcal{M}_i\sim p(\mathcal{M})}\left[\mathcal{L}_{\mathcal{M}_i}(f_{\theta'_i})\right] = \min_\theta \mathbb{E}_{\mathcal{M}_i\sim p(\mathcal{M})}\left[\mathcal{L}_{\mathcal{M}_i}(f_{\theta-\alpha\nabla_\theta\mathcal{L}_{\mathcal{M}_i}(f_\theta)})\right] \tag{7.2}$$

After meta-training, to quickly adapt to a new test environment, MAML computes a task-specific loss function from an unseen task and updates θ with several gradient steps.

7.4 Problem Statement

We consider the offline meta-reinforcement learning problem setting introduced in [154, 155], in which the objective is to leverage offline data from a number of different tasks to rapidly adapt to an unseen task at test-time. We consider an instantiation of this setting in which tasks correspond to CMDPs $\{\mathcal{M}_i\}_{i=1}^N$, each with different system dynamics $p_i(s'|s,a)$, but which otherwise share all other MDP parameters, including the same state and action spaces and constraint function. Here the agent is not allowed to directly interact with any environment at meta-train time or meta-test time, but is only provided with a fixed offline dataset of transitions from environments. This setting is particularly applicable to the safe reinforcement learning setting, where direct environmental interaction can be risky, but there may be accident logs from prior robot deployments in various settings. We formalize the problem of learning about constraints in the environment in the context of offline meta-reinforcement learning, in which the agent is provided with offline data from N_{train} training environments $\{\mathcal{M}_i^{\text{train}}\}_{i=1}^{N_{\text{train}}}$ with varying system dynamics and must rapidly adapt to being safe in a new environment $\mathcal{M}^{\text{test}}$ with unseen system dynamics. The intuition is that when dynamics change, the states which violate constraints remain the same, but the behaviors that lead to these states may be very different. Thus, we consider the problem of using data from a number of training environments to optimize the safe RL objective in Equation 7.1.

We assume that the agent is provided with a set of N_{train} datasets of offline transitions $\mathcal{D}^{\text{train}} = \{\mathcal{D}_i^{\text{train}}\}_{i=1}^{N_{\text{train}}}$ from training environments with different dynamics in addition to a small dataset $\mathcal{D}^{\text{test}}$ of offline transitions from the test environment $\mathcal{M}^{\text{test}}$, in which the agent is to be deployed. The agent's objective is to leverage this data to optimize the safe RL objective in Equation 7.1 in MDP $\mathcal{M}^{\text{test}}$ by learning some task τ in MDP $\mathcal{M}^{\text{test}}$ while minimizing constraint violations.

7.5 MEta-learning for Safe Adaptation (MESA)

We introduce MEta-learning for Safe Adaptation (MESA), a 3-phase procedure to optimize the objective in Section 7.4. First, MESA uses datasets of offline transitions from the training environments to meta-learn a safety critic optimized for rapid adaptation (Section 7.5.1). Then, we discuss how MESA adapts its meta-learned safety critic using a dataset of offline transitions from the test environment (Section 7.5.2). This same dataset is also used to learn a recovery policy, which is trained to descend the safety critic and prevent the agent from visiting unsafe states as in Thananjeyan et al. [8], but we note that the learned safety critic can also be used in conjunction with other safe RL algorithms such as those from Bharadhwaj et al. [4] and Srinivasan et al. [145]. Finally, the meta-learned safety critic and recovery policy are used and updated online when learning some downstream task τ in the testing environment (Section 7.5.3). The full algorithm is summarized in Algorithm 5 and Figure 7.2. An illustration of the safety critic adaptation procedure is shown in Figure 7.1.

7.5.1 Phase 1, Meta-Learning Q^{π}_{risk}

Given offline transitions from N_{train} training environments, $\{\mathcal{D}_i^{\text{train}}\}_{i=1}^{N_{\text{train}}}$, we meta-learn the safety critic $Q^{\pi}_{\psi,\text{risk}}$, with parameters ψ, using Model-agnostic Meta Learning [172]. We

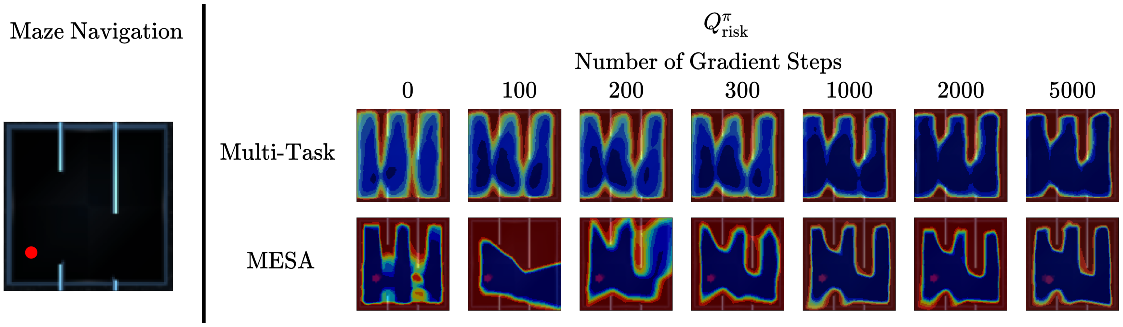


Figure 7.1: **Safety Critic Adaptation Visualizations:** For purposes of illustration, we evaluate MESA and a Multi-Task learning comparison on a simple Maze Navigation task (left) from [8] in which the objective is for the agent (the red dot) to navigate from a random point in the left column to the middle of the right column without colliding into any of the Maze walls or boundaries. Environments are sampled by changing the gaps in the walls (parameterized by $w_1, w_2 \sim \mathcal{U}(-0.1, 0.1)$), leading to significant changes in which behaviors are safe. On the left, we show heatmaps of the learned safety critic Q^{π}_{risk} when it is adapted to a new Maze with unseen wall gaps for the Multi-Task comparison (top) and MESA (bottom). Here bluer colors denote low probability of constraint violation while redder colors denote a higher probability, and the labels above the heatmaps indicate the number of gradient steps used for adaptation on $\mathcal{D}^{\text{test}}$. The Multi-Task learning comparison, which aggregates data from all environments to learn the safety critic and does not explicitly optimize for adaptation, is much slower to adapt while MESA is able to leverage its learned prior to rapidly adapt to the new gap positions.

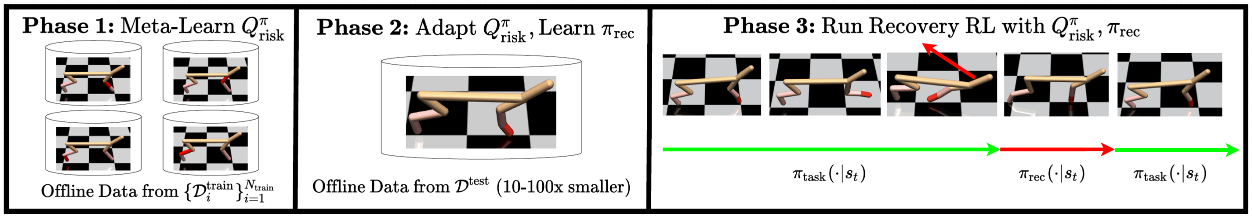
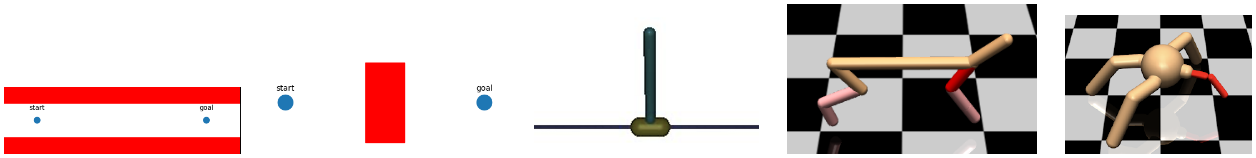

Phase 1: Meta-Learn Q^{π}_{risk}
Offline Data from $\{\mathcal{D}^{\text{train}}_i\}_{i=1}^{N_{\text{train}}}$
Phase 2: Adapt Q^{π}_{risk}, Learn π_{rec}
Offline Data from $\mathcal{D}^{\text{test}}$ (10-100x smaller)
Phase 3: Run Recovery RL with Q^{π}_{risk}, π_{rec}
$\pi_{\text{task}}(\cdot|s_t)$
$\pi_{\text{rec}}(\cdot|s_t)$
$\pi_{\text{task}}(\cdot|s_t)$

Figure 7.2: **Left: MESA:** MESA takes a 3 phase approach to learn a transferable risk measure for safe RL. In Phase 1, MESA uses offline datasets from training environments of different dynamics to meta-learn a safety critic Q^{π}_{risk}. In Phase 2, MESA adapts the safety critic to a test environment with unseen dynamics using a small test dataset. Finally, in Phase 3, MESA uses the adapted safety critic and recovery policy in the test environment to enable safe learning as in Recovery RL.

utilize the same safety critic loss function from [8]. The recovery policy is not trained with a MAML-style objective. Similar to the actor's loss function in DDPG [135], the recovery policy, parameterized by ω, aims to minimize the safety critic value for input state s_t:

$$\mathcal{L}_{\pi_{\text{rec}}}(\omega, s_t) = Q^{\pi}_{\psi,\text{risk}}(s_t, \pi_{\omega,\text{rec}}(\cdot|s_t)).$$



(a) Navigation 1 (b) Navigation 2 (c) Cartpole Length (d) HalfCheetah Disabled (e) Ant Disabled

Figure 7.3: **Simulation Domains:** We evaluate MESA on a set of 2D navigation and locomotion tasks in simulation. In Navigation 1 and Navigation 2, the agent learns to navigate from a beginning position to the goal while avoiding the obstacles (red walls). In the Cartpole-Length task, the goal is to keep the pole balanced on the cart while minimizing the number of times the pole falls beneath the rail or moves off the rail. Lastly, in the HalfCheetah-Disabled and Ant-Disabled tasks, the objective is to learn how to move forwards while minimizing the number of collisions with the ground of the head (HalfCheetah) or torso (Ant) during training.

7.5.2 Phase 2, Test Time Adaptation

A previously unseen test environment $\mathcal{M}^{\text{test}}$ is sampled from task distribution $p(\mathcal{M})$ and the agent is supplied with a dataset of offline transitions $\mathcal{D}_{\text{test}}$, which is **10-100x** smaller than the training datasets. We then perform M gradient steps with respect to $\mathcal{L}_{\text{risk}}(\psi, s)$ (in Section 7.3.2) and $\mathcal{L}_{\pi_{\text{rec}}}(\omega, s)$ over $\mathcal{D}_{\text{test}}$ to rapidly adapt safety critic $Q^{\pi}_{\psi,\text{risk}}$ and train recovery policy $\pi_{\omega,\text{rec}}$

Note that the learned $Q^{\pi}_{\psi,\text{risk}}$ is initially calibrated with the policy used for data collection in the meta-training environments. Since these datasets largely consist of constraint violations, the resulting $Q^{\pi}_{\psi,\text{risk}}$ serves as a pessimistic initialization for online learning of some downstream task τ. This is a desirable property, as $Q^{\pi}_{\psi,\text{risk}}$ will initially prevent constraint violations, and then become increasingly less pessimistic during online exploration when calibrated with the task policy for task τ.

7.5.3 Phase 3, Using Q^{π}_{risk} and π_{rec} for Safe RL

We initialize the safety critic and recovery policy with the adapted $Q^{\pi}_{\psi,\text{risk}}$ and $\pi_{\omega,\text{rec}}$ when learning a task τ in the test environment. Since the safety critic is learned offline in a task-agnostic way, we can flexibly utilize the meta-learned safety critic and recovery policy to learn a previously unknown task τ in the test environment. As in Recovery RL [8], both $Q^{\pi}_{\psi,\text{risk}}$ and $\pi_{\omega,\text{rec}}$ are updated online through interaction with the environment so that they are calibrated with the learned task policy for τ.

7.6 Experiments

We study the degree to which MESA can leverage offline data from environments with different dynamics to quickly learn safety in a new test domain with modified, previously unseen dynamics via a small amount of experience in the new domain. To do this, we compare MESA with prior safe reinforcement learning algorithms and study the degree to which they can limit constraint violations when learning in a perturbed test environment with previously unseen dynamics. MEta-learning for Safe Adaptation (MESA) and all comparisons are built on top of the Soft Actor Critic (SAC) algorithm from Haarnoja et al. [62]. **Comparisons:** We compare MESA with the following algorithms: **Unconstrained:** A soft actor critic agent which only optimizes for task rewards and ignores constraints; **Recovery RL (RRL):** Uses data only from $\mathcal{D}_{\text{test}}$ to learn Q^{π}_{risk} and then uses Q^{π}_{risk} in conjunction with the Recovery RL algorithm [8]; **Multi-Task Learning (Multi-Task):** Learns Q^{π}_{risk} from a combination of all data from both the training datasets $\{\mathcal{D}_i\}_{i=1}^{N_{\text{train}}}$ in phase 1 and then adapts in phase 2 using gradient steps on only the test dataset $\mathcal{D}_{\text{test}}$. In phase 3, Multi-Task uses the learned Q^{π}_{risk} in conjunction with the Recovery RL algorithm [8] as in MESA and the RRL comparison; **CARL:** A prior safe meta-reinforcement learning algorithm which learns a dynamics model and safety indicator function through interaction with number of source environments and uses the uncertainty of the learned dynamics models to adapt to a target environment with previously unknown dynamics in a risk-averse manner; **CARL-Offline:** A modification of CARL which only provides CARL with offline datasets from the source environments, consistent with the offline meta-RL setting we consider in this chapter.

The comparison to Unconstrained allows us to evaluate the effect of reasoning about constraints at all. The comparison to Recovery RL allows us to understand whether offline data from different environments enables MESA to learn about constraints in the test environment. The comparison to the Multi-Task Learning algorithm allows us to evaluate the benefits of specifically leveraging meta-learning to quickly adapt learned risk measures. The comparisons to CARL and CARL-Offline allow us to evaluate whether MESA can outperform prior work in safe meta-RL.

Experimental Procedure: We evaluate MESA and comparisons on their ability to (1) efficiently learn some downstream task τ in the test environment (2) while satisfying constraints. We report learning curves and cumulative constraint violations for all algorithms to see if MESA can leverage prior experience to safely adapt in the test environment. Episodes are terminated upon a constraint violation, making learning about constraints critical for safely learning in the test environment. We report average performance over 5 random seeds with standard error shading for all learning curves.

Domains: We evaluate MESA and comparisons on 5 simulation domains which are illustrated in Figure 7.3. All domains we study have the property that the changes in the dynamics are not immediately observable in the agent's observation, motivating learning how to be safe from interaction experience when dynamics change. This is common in various practical settings, such as a robot with worn out joints or sudden power loss in a legged locomotion system. We first consider two 2D navigation domains from [8] in which the agent must navigate between a start set and goal set without colliding into red obstacles in a system with linear Gaussian dynamics. The environment distribution for both domains is defined by varying the coefficients of the A and B matrices in the transition dynamics function where $s_{t+1} = A \cdot s_t + B \cdot a_t + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$.

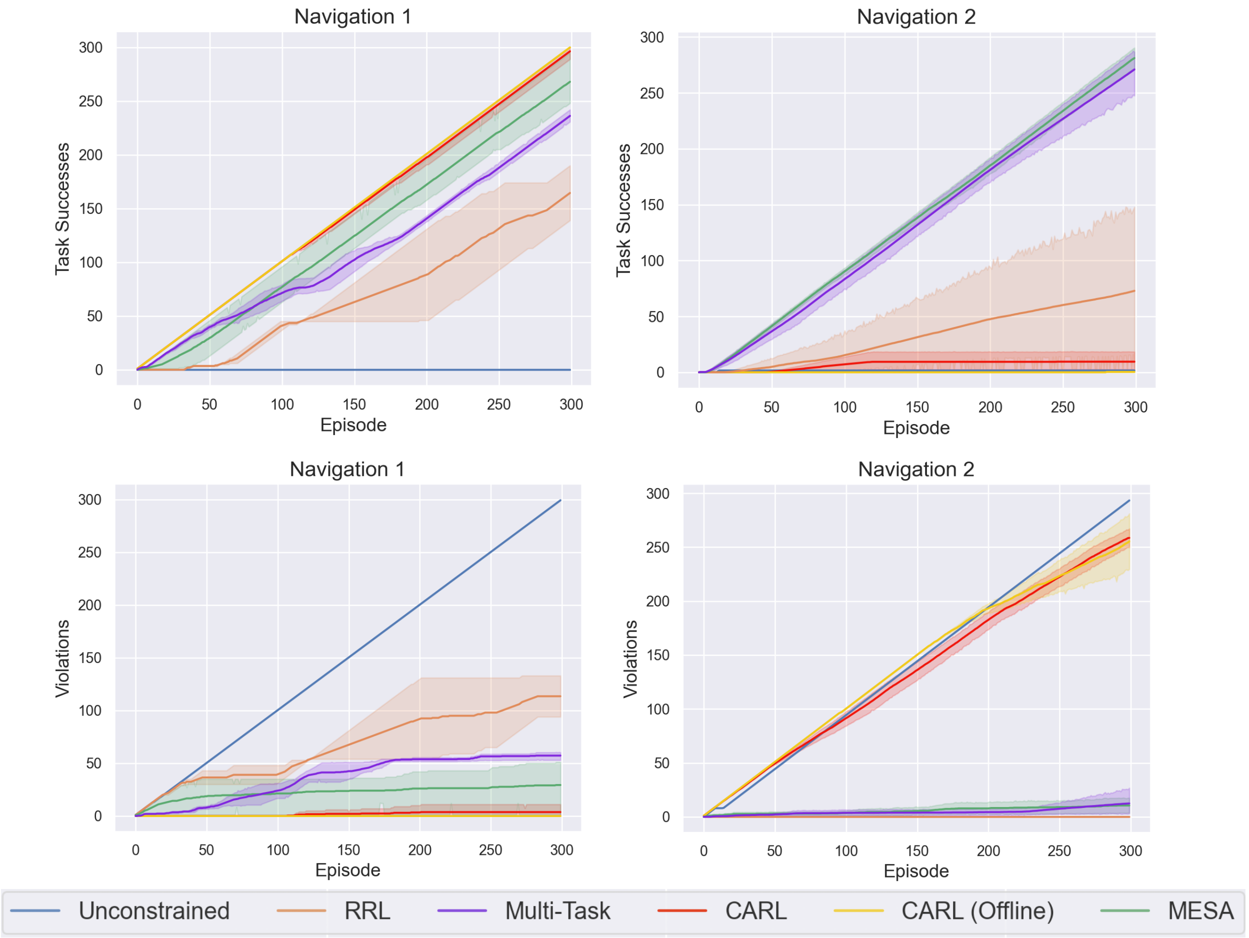


Figure 7.4: **Navigation Results: Top: Learning Curves (Phase 3).** MESA is able to achieve similar task success compared to prior algorithms on bot domains. **Bottom: Cumulative Constraint Violations (Phase 3).** Here, we find that MESA achieves fewer constraint violations than most comparisons, but find that the Multi-Task comparison also performs well on these environments.

We then consider a cartpole task (Cartpole-Length) in which the agent must balance the cartpole system without letting the pole fall below the cart. Here environments are sampled by varying the length of the pole, where pole lengths for the training environments are sampled from $\mathcal{U}(0.4, 0.8)$ and the test environment corresponds to a pole of length 1. We also consider two legged locomotion tasks, HalfCheetah-Disabled and Ant-Disabled, in which the agent is rewarded for running as fast as possible, but violates constraints given a collision of the head with the floor or torso with the floor for the HalfCheetah-Disabled and Ant-Disabled tasks respectively. For both HalfCheetah-Disabled and Ant-Disabled, environments are sampled by choosing a specific joint and simulating a loss of power (power loss corresponds to always providing zero motor torque to the joint), resulting in significantly different dynamics across environments. The Cartpole-Length and HalfCheetah-Disabled tasks are adapted from [108] while the Ant-Disabled task is from [26].

7.6.1 Data Collection

For the navigation environments, offline datasets are collected via a random policy where the episode *does not terminate* upon constraint violation. We collect a total of 20-25 datasets for each of the sampled training environments, with each dataset consisting of 10000 transitions (680 and 1200 violations in Navigation 1 and Navigation 2 respectively), similar to that of [8]. However, the dataset in the test environment is **50-100x** smaller than each training task dataset (∼100, 200 transitions with 15, 36 violations respectively).

Similarly, for locomotion environments, the datasets from the test environment are collected via a random policy rollout, where the episode does not terminate early upon constraint violations. To collect datasets from the training environments, we train SAC on each of the training environments and log the replay buffer from an intermediate checkpoint. For the HalfCheetah-Disabled and Ant-Disabled tasks, we collect 4 and 3 training datasets of 400 episodes (on average ∼400K transitions with 14K and 113K violations) respectively. The dataset from the testing environment consists of 40K transitions (2.4K, and 11.2K violations for HalfCheetah, Ant), which is **10x** smaller than before. For the Cartpole-Length task, 20 training datasets are generated, with each containing 200 episodes of data (∼20K timesteps with 4.5K violations). The dataset from the testing environment contains 1K transitions (with 200 violations), which is **20x** smaller than before.

7.6.2 Results

Navigation Results: We evaluate the performance of MESA and comparisons in Figure 7.4. Unconstrained SAC performs poorly as it no mechanism to reason about constraints and thus collides frequently and is unable to learn the task. MESA violates constraints less often than the Multi-Task comparison, but the performance gap is somewhat small in these environments. We hypothesize that this is because in the Navigation environments, particularly Navigation 2, the space of safe behaviors does not change significantly as a function of the system dynamics, making it possible for the Multi-Task comparison to achieve strong perfor-

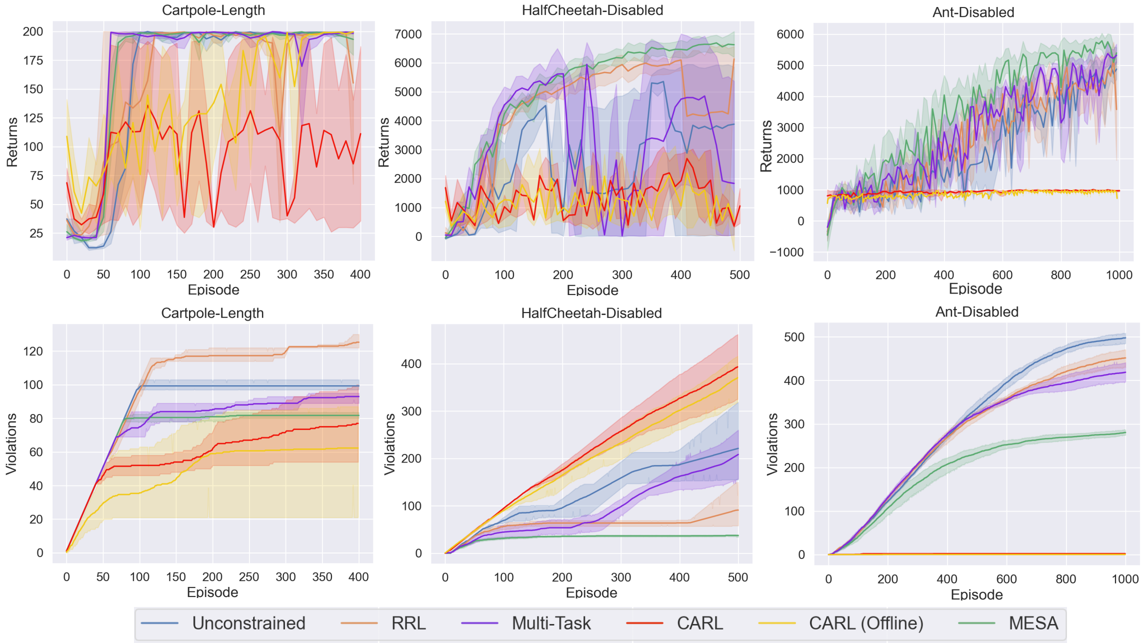
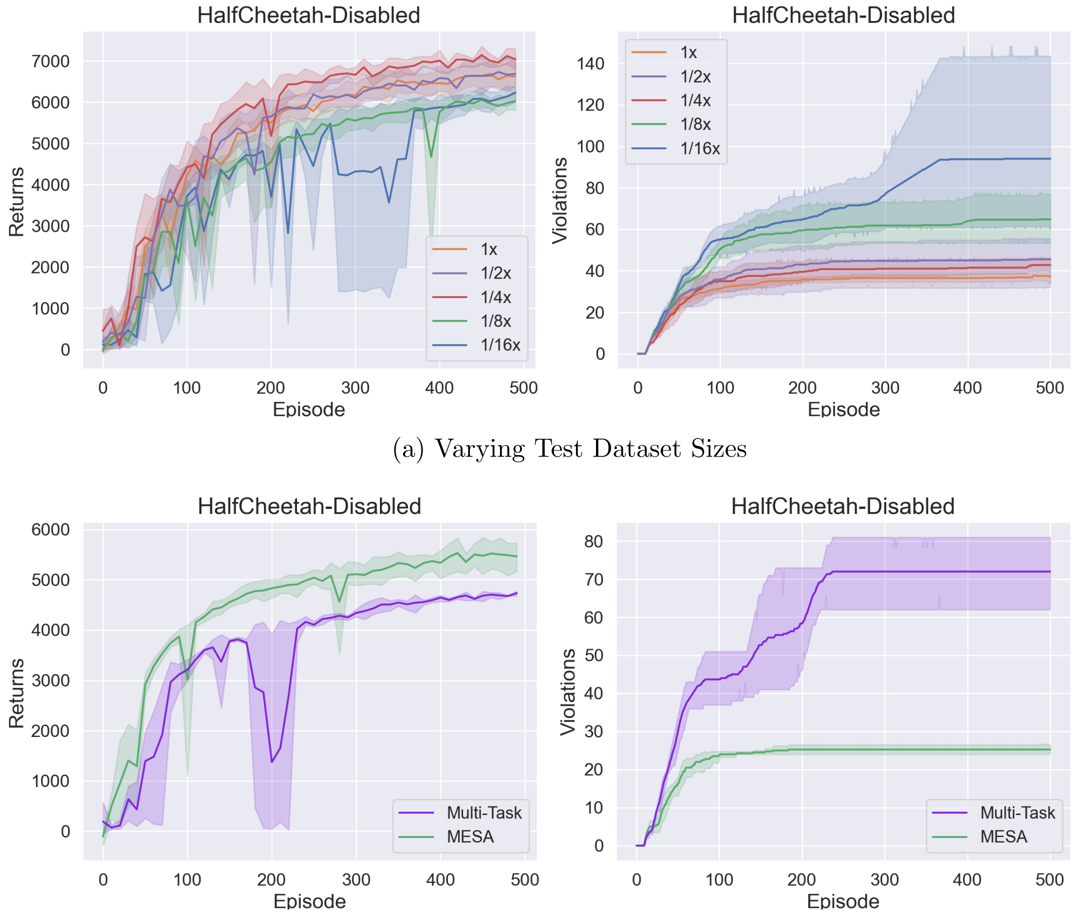


Figure 7.5: **Locomotion Results: Top: Learning Curves (Phase 3).** MESA achieves similar task performance as the best comparison algorithm, indicating that MESA is able to effectively learn in a test environment with previously unseen dynamics. **Bottom: Cumulative Constraint Violations (Phase 3).** MESA violates constraints less often than comparisons, with this difference being most significant on the HalfCheetah-Disabled and Ant-Disabled tasks. This suggests that MESA is able to effectively leverage its prior experiences across environments with different dynamics to rapidly adapt its risk measure to the test environment.

mance by simply learning the safety critic jointly on a buffer of all collected data. CARL and CARL-Offline baselines perform the best in the Navigation 1 environment but are unable to make much progress in Navigation 2.

Locomotion Results: MESA significantly outperforms prior methods on the HalfCheetah-Disabled and Ant-Disabled, while achieving comparable performance on the Cartpole task (Figure 7.5). We hypothesize that in the HalfCheetah-Disabled and Ant-Disabled tasks, the different training environments are sufficiently different in their dynamics that a safety critic and recovery policy trained jointly on all of them is unable to accurately represent the boundaries between safe and unsafe states. Thus, when adapting to an environment with unseen dynamics, the space of safe behaviors may be so different than in the training environments that the Multi-Task comparison cannot easily adapt. MESA mitigates this by explicitly optimizing the safety critic for rapid adaptation to different dynamics. In addition, CARL and CARL-Offline make little task progress in the HalfCheetah and Ant Disabled domain and, as a result, are able to generally satisfy constraints. The sharp decline in performance is likely due to the planning algorithm that CARL utilizes for optimization over learned dynamics.



(a) Varying Test Dataset Sizes

(b) Test Task Generalization: Partial Joint Failures

Figure 7.6: **Ablation: Sensitivity to Test Dataset Size:** In Figure 7.6a, we investigate the sensitivity of MESA to the number of transitions in the test dataset used for adapting Q^{π}_{risk} for HalfCheetah-Disabled. We find that even with a test dataset 4 times smaller than used in the experiments in Section 7.6, MESA does not experience much degradation in performance. However, further reductions in the test dataset size make it difficult for MESA to learn a sufficiently accurate safety critic in the test environment, leading to more significant drops in performance. **Generalization to More Different Test Environment Dynamics:** In Figure 7.6b, we investigate MESA's and Multi-Task's generalization to partial joint failures in the HalfCheetah-Disabled task, where the training sets are kept the same as described in Section 7.6. We find that MESA is able to significantly reduce the number of constraint violations compared to the Multi-Task comparison while also achieving superior task performance, suggesting that as differences in system dynamics increase between the training and testing environments, MESA is able to more effectively adapt risk measures across the environments.

7.7 Ablations

In ablations, we seek to answer the following questions: (1) how small can the dataset from the test environment be for MESA to safely adapt to new test environments? and (2) how well can MESA generalize to environments consisting of more significantly different dynamics (e.g. partial joint failures when only trained on datasets with examples of full joint failures)?

7.7.1 Test Dataset Size

We first investigate the sensitivty of MESA to the size of the test dataset. Figure 7.6a, we study performance when the test dataset is 1x, 1/2x, 1/4x, 1/8x, and 1/16x the size of the test dataset (40K transitions) used for the HalfCheetah-Disabled results reported in Section 7.6. We find that MESA can do well when given a test dataset 1/4 the size of the original test dataset (10K transitions, which is 10 episodes of environment interaction). This suggests that the test size dataset can be up to **40x** smaller than the training dataset sizes without significant drop in performance. We find that when the test dataset is reduced to 1/8 and 1/16 the size of the original test dataset, MESA exhibits degrading performance, as the safety critic has insufficient data to learn about constraints in the test environment.

7.7.2 Test Environment Generalization

Here we study how MESA performs when the test environments have more significantly different dynamics from those seen during training. To evaluate this, we consider the HalfCheetah-Disabled task, and train MESA using the same training datasets considered in Section 7.6, in which specific joints are selected to lose power. However, at test time, we evaluate MESA on a setting with partial power losses to joints, in which the maximum applicable power to certain joints is set to some k percent of the original maximum power, where $k \in \mathcal{U}(0.5, 0.95)$. This is analogous to partial subsystem failures that can occur in real-world robotic systems. In, Figure 7.6b, we find that MESA achieves superior performance compared to the the Multi-Task comparison in terms of both task performance and constraint violations during training. This suggests that MESA could rapidly learn to be safe even with system dynamics that are out of the meta-training environment distribution.

7.8 Conclusion

We formulate safe reinforcement learning as an offline meta-reinforcement learning problem and motivate how learning from offline datasets of unsafe behaviors in previous environments can provide a scalable and compelling way to learn tasks safely in new environments with unobserved change in system dynamics. We then present MEta-learning for Safe Adaptation (MESA), a new algorithm for learning a risk measure which can transfer knowledge about safety across environments with different dynamics. Results in simulation experiments suggest that MESA is able to achieve strong performance across 5 different robotic simulation domains and is able to effectively adapt to test environments with previously unseen dynamics.

Part III

Safe Exploration Using Policy Uncertainty

Chapter 8

Querying a Human Recovery Policy Based on Task Uncertainty

8.1 Introduction

Imitation learning allows a robot to learn from human feedback and examples [185–187]. In particular, *interactive imitation learning* (IL) [18, 19, 188], in which a human supervisor periodically takes control of the robotic system during policy learning, has emerged as a popular imitation learning method, as interventions are a particularly intuitive form of human feedback [188]. However, a key challenge in interactive imitation learning is to reduce the burden that interventions place on the human supervisor [18, 19].

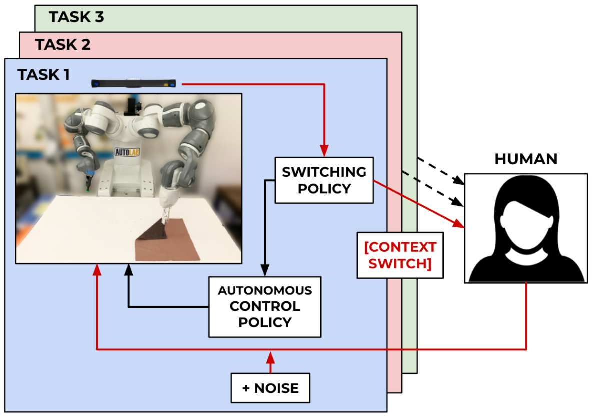


Figure 8.1: LazyDAgger learns to cede control to a supervisor in states in which it estimates that its actions will significantly deviate from those of the supervisor. LazyDAgger reduces context switches between supervisor and autonomous control to reduce burden on a human supervisor working on multiple tasks.

One source of this burden is the cost of *context switches* between human and robot control. Context switches incur significant time cost, as a human must interrupt the task they are currently performing, acquire control of the robot, and gain sufficient situational awareness before beginning the intervention. As an illustrative example, consider a robot performing a task for which an action takes 1 time unit and an intervention requires two context switches (one at the start and one at the end). We define *latency* L as the number of time units associated with a single context switch. For instance, $L \gg 1$ for a human supervisor who will need to pause an ongoing task and walk over to a robot that requires assistance. If the supervisor takes control 10 times for 2 actions each, she spends $20L + 20$ time units helping the robot. In contrast, if the human takes control 2 times for 10 actions each, she spends only $4L + 20$ time units. The latter significantly reduces the burden on the supervisor. Furthermore, prior work suggests that frequent context switches can make it difficult for the supervisor to perform other tasks in parallel [189] or gain enough situational awareness to provide useful interventions [190].

We present LazyDAgger (Figure 8.1), an algorithm which initiates useful interventions while limiting context switches. The name LazyDAgger is inspired by the concept of lazy evaluation in programming language theory [191], where expressions are evaluated only when required to reduce computational burden. As in SafeDAgger [19], LazyDAgger learns a meta-controller which determines when to context switch based on the estimated discrepancy between the learner and supervisor. However, unlike SafeDAgger, LazyDAgger reduces context switching by (1) introducing asymmetric switching criteria and (2) injecting noise into the supervisor control actions to widen the distribution of visited states. One appealing property of this improved meta-controller is that even after training, LazyDAgger can be applied at execution time to improve the safety and reliability of autonomous policies with minimal context switching. We find that across 3 continuous control tasks in simulation, LazyDAgger achieves task performance on par with DAgger [122] with 88% fewer supervisor actions than DAgger and 60% fewer context switches than SafeDAgger. In physical fabric manipulation experiments, we observe similar results, and find that at execution time, LazyDAgger achieves 60% better task performance than SafeDAgger with 60% fewer context switches.

8.2 Background and Related Work

Challenges in learning efficiency and reward function specification have inspired significant interest in algorithms that can leverage supervisor demonstrations and feedback for policy learning.

Learning from Offline Demonstrations: Learning from demonstrations [185–187] is a popular imitation learning approach, as it requires minimal supervisor burden: the supervisor provides a batch of offline demonstrations and gives no further input during policy learning. Many methods use demonstrations directly for policy learning [192–195], while others use reinforcement learning to train a policy using a reward function inferred from demonstrations [196–200]. Recent work has augmented demonstrations with additional offline information such as pairwise preferences [69, 201], human gaze [202], and natural language descriptions [203]. While offline demonstrations are often simple to provide, the lack of online feedback makes it difficult to address specific bottlenecks in the learning process or errors in the resulting policy due to covariate shift [122].

Learning from Online Feedback: Many policy learning algorithms' poor performance stems from a lack of online supervisor guidance, motivating active learning methods such as DAgger, which queries the supervisor for an action in every state that the learner visits [122]. While DAgger has a number of desirable theoretical properties, labeling every state is costly in human time and can be a non-intuitive form of human feedback [204]. More generally, the idea of learning from action advice has been widely explored in imitation learning algorithms [205–208]. There has also been significant recent interest in active preference queries for learning reward functions from pairwise preferences over demonstrations [201, 209–213]. However, many forms of human advice can be unintuitive, since the learner may visit states that are significantly far from those the human supervisor would visit, making it difficult for humans to judge what correct behavior looks like without interacting with the environment themselves [188, 214].

Learning from Supervisor Interventions: There has been significant prior work on algorithms for learning policies from interventions. Xie et al. [215] and Kurenkov et al. [216] leverage interventions from suboptimal supervisors to accelerate policy learning, but assume that the supervisors are algorithmic and thus can be queried cheaply. Thananjeyan et al. [8], Nolan Wagener [217], and Saunders et al. [218] also leverage interventions from algorithmic policies, but for constraint satisfaction during learning. Kelly et al. [18], Mandlekar et al. [21], Spencer et al. [188], Wang et al. [219], Kahn et al. [220], and Amir et al. [221] instead consider learning from human supervisors and present learning algorithms which utilize the timing and nature of human interventions to update the learned policy. By giving the human control for multiple timesteps in a row, these algorithms show improvements over methods that only hand over control on a state-by-state basis [222]. However, the above algorithms assume that the human is continuously monitoring the system to determine when to intervene, which may not be practical in large-scale systems or continuous learning settings [189, 223–225]. Such algorithms also assume that the human knows when to cede control to the robot, which requires guessing how the robot will behave in the future. Zhang et al. [19] and Menda et al. [20] present imitation learning algorithms SafeDAgger and EnsembleDAgger, respectively, to address these issues by learning to request interventions from a supervisor based on measures such as state novelty or estimated discrepancy between the learner and supervisor actions. These methods can still be sample inefficient, and, as we discuss later, often result in significant context switching.

By contrast, LazyDAgger encourages interventions that are both easier to provide and more informative. To do this, LazyDAgger prioritizes (1) sustained interventions, which allow the supervisor to act over a small number of contiguous sequences of states rather than a large number of disconnected intervals, and (2) interventions which demonstrate supervisor actions in novel states to increase robustness to covariate shift in the learned policy.

8.3 Problem Statement

We consider a setting in which a human supervisor is training a robot to reliably perform a task. The robot may query the human for assistance, upon which the supervisor takes control and teleoperates the robot until the system determines that it no longer needs assistance. We assume that the robot and human policy have the same action space, and that it is possible to pause task execution while waiting to transfer control. We formalize these ideas in the context of prior imitation learning literature.

We model the environment as a discrete-time Markov decision process (MDP) $\mathcal{M}$ with states $s \in \mathcal{S}$, actions $a \in \mathcal{A}$, and time horizon T [226]. The robot does not have access to the reward function or transition dynamics of $\mathcal{M}$ but can cede control to a human supervisor, who executes some deterministic policy $\pi_H : \mathcal{S} \to \mathcal{A}$. We refer to times when the robot is in control as *autonomous mode* and those in which the supervisor is in control as *supervisor mode*. We minimize a surrogate loss function $J(\pi_R)$ to encourage the robot policy $\pi_R : \mathcal{S} \to \mathcal{A}$ to match that of the supervisor (π_H):

$$J(\pi_R) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi_R}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right], \tag{8.1}$$

where $\mathcal{L}(\pi_R(s), \pi_H(s))$ is an action discrepancy measure between $\pi_R(s)$ and $\pi_H(s)$ (e.g., the squared loss or 0-1 loss), and $d_t^{\pi_R}$ is the marginal state distribution at timestep t induced by executing π_R in MDP $\mathcal{M}$.

In interactive IL we require a meta-controller π that determines whether to query the robot policy π_R or to query for an intervention from the human supervisor policy π_H; importantly, π consists of both (1) the high-level controller which decides whether to switch between π_R and π_H and (2) the low-level robot policy π_R. A key objective in interactive IL is to minimize some notion of supervisor burden. To this end, let $m_I(s_t; \pi)$ be an indicator which records whether a context switch between autonomous (π_R) and supervisor (π_H) modes occurs at state s_t (either direction). Then, we define $C(\pi)$, the expected number of context switches in an episode under policy π, as follows: $C(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi}} [m_I(s_t; \pi)]$, where d_t^{π} is the marginal state distribution at timestep t induced by executing the meta-controller π in MDP $\mathcal{M}$. Similarly, let $m_H(s_t; \pi)$ indicate whether the system is in supervisor mode at state s_t. We then define $D(\pi)$, the expected number of supervisor actions in an episode for the policy π, as follows: $D(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi}} [m_H(s_t; \pi)]$.

We define *supervisor burden* $B(\pi)$ as the expected time cost imposed on the human supervisor. This can be expressed as the sum of the expected total number of time units spent in context switching and the expected total number of time units in which the supervisor is actually engaged in performing interventions:

$$B(\pi) = L \cdot C(\pi) + D(\pi), \tag{8.2}$$

where L is context switch latency in time units, and each time unit is the time it takes for the supervisor to execute a single action. The learning objective is to find a policy π

that matches supervisor performance, π_H, while limiting supervisor burden to lie within a threshold Γ_b, set by the supervisor to an acceptable tolerance for a given task. To formalize this problem, we propose the following objective:

$$\pi = \underset{\pi' \in \Pi}{\arg\min} \{ J(\pi'_R) \mid B(\pi') \leq \Gamma_b \}, \tag{8.3}$$

where Π is the space of all meta-controllers, and π'_R is the low-level robot policy associated with meta-controller π'.

8.4 Preliminaries: SafeDAgger

We consider interactive IL in the context of the objective introduced in Equation (8.3): to maximize task reward while limiting supervisor burden. To do this, LazyDAgger builds on SafeDAgger [19], a state-of-the-art algorithm for interactive IL. SafeDAgger selects between autonomous mode and supervisor mode by training a binary action discrepancy classifier f to discriminate between "safe" states which have an action discrepancy below a threshold β_H (i.e., states with $\mathcal{L}(\pi_R(s), \pi_H(s)) < \beta_H$) and "unsafe" states (i.e. states with $\mathcal{L}(\pi_R(s), \pi_H(s)) \geq \beta_H$). The classifier f is a neural network with a sigmoid output layer (i.e., $f(s) \in [0, 1]$) that is trained to minimize binary cross-entropy (BCE) loss on the datapoints $(s_t, \pi_H(s_t))$ sampled from a dataset $\mathcal{D}$ of trajectories collected from π_H. This is written as follows:

$$\begin{aligned} \mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f) = &-f^*(\pi_R(s_t), \pi_H(s_t)) \log f(s_t) \\ &-(1 - f^*(\pi_R(s_t), \pi_H(s_t))) \log(1 - f(s_t)), \end{aligned} \tag{8.4}$$

where the training labels are given by $f^*(\pi_R(s_t), \pi_H(s_t)) = \mathbb{1}\{\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \geq \beta_H\}$, and $\mathbb{1}$ denotes the indicator function. Thus, $\mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f)$ penalizes incorrectly classifying a "safe" state as "unsafe" and vice versa.

SafeDAgger executes the meta-policy π which selects between π_R and π_H as follows:

$$\pi(s_t) = \begin{cases} \pi_R(s_t) \text{ if } f(s_t) < 0.5 \\ \pi_H(s_t) \text{ otherwise,} \end{cases} \tag{8.5}$$

where $f(s_t) < 0.5$ corresponds to a prediction that $\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) < \beta_H$, i.e., that s_t is "safe." Intuitively, SafeDAgger only solicits supervisor actions when f predicts that the action discrepancy between π_R and π_H exceeds the safety threshold β_H. Thus, SafeDAgger provides a mechanism for querying the supervisor for interventions only when necessary. In LazyDAgger, we utilize this same mechanism to query for interventions but enforce new properties once we enter these interventions to lengthen them and increase the diversity of states observed during the interventions.

8.5 LazyDAgger

We summarize LazyDAgger in Algorithm 1. In the initial phase (Lines 1-3), we train π_R and safety classifier f on offline datasets collected from the supervisor policy π_H. In the interactive learning phase (Lines 4-19), we evaluate and update the robot policy for N epochs, ceding control to the supervisor when the robot predicts a high action discrepancy.

8.5.1 Action Discrepancy Prediction

SafeDAgger uses the classifier f to select between π_R and π_H (Equation (8.5)). However, in practice, this often leads to frequent context switching (Figure 8.3). To mitigate this, we make two observations. First, we can leverage that in supervisor mode, we directly observe the supervisor's actions. Thus, there is no need to use f, which may have approximation errors, to determine whether to remain in supervisor mode; instead, we can compute the ground-truth action discrepancy $\mathcal{L}(\pi_R(s_t), \pi_H(s_t))$ exactly for any state s_t visited in supervisor mode by comparing the supplied supervisor action $\pi_H(s_t)$ with the action proposed by the robot policy $\pi_R(s_t)$. In contrast, SafeDAgger uses f to determine when to switch modes both in autonomous and supervisor mode, which can lead to very short interventions when f prematurely predicts that the agent can match the supervisor's actions. Second, to ensure the robot has returned to the supervisor's distribution, the robot should only switch

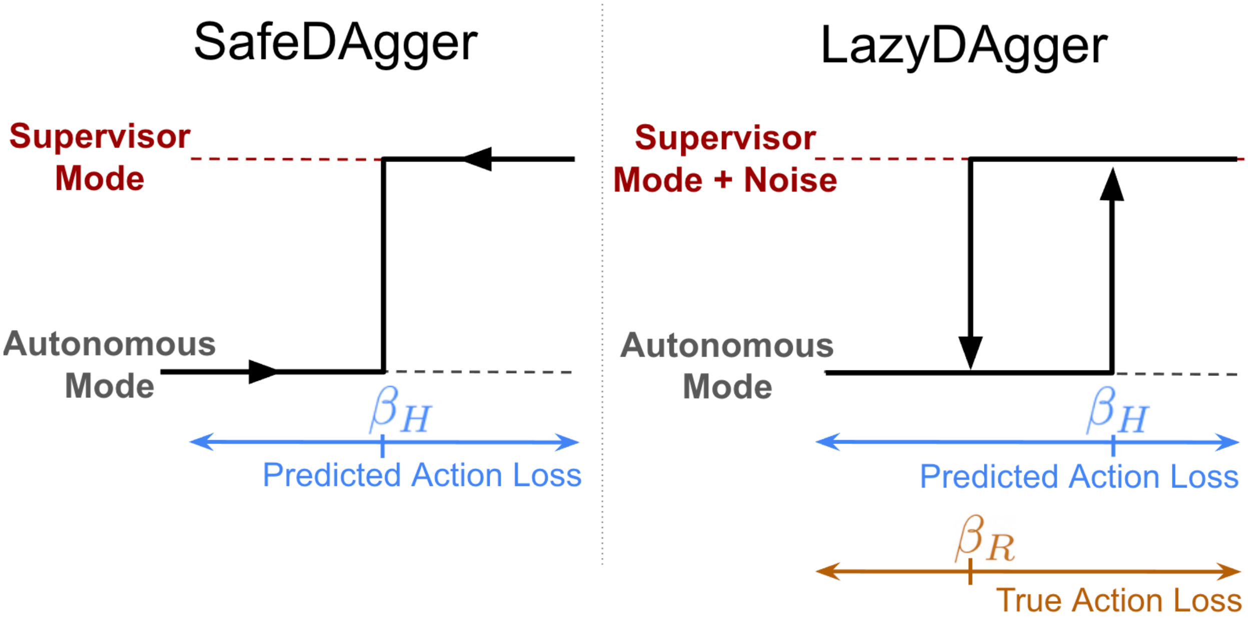


Figure 8.2: **LazyDAgger Switching Strategy:** SafeDAgger switches between supervisor and autonomous mode if the predicted action discrepancy is above threshold β_H. In contrast, LazyDAgger uses asymmetric switching criteria and switches to autonomous mode based on ground truth action discrepancy. The gap between β_R and β_H defines a hysteresis band [227].

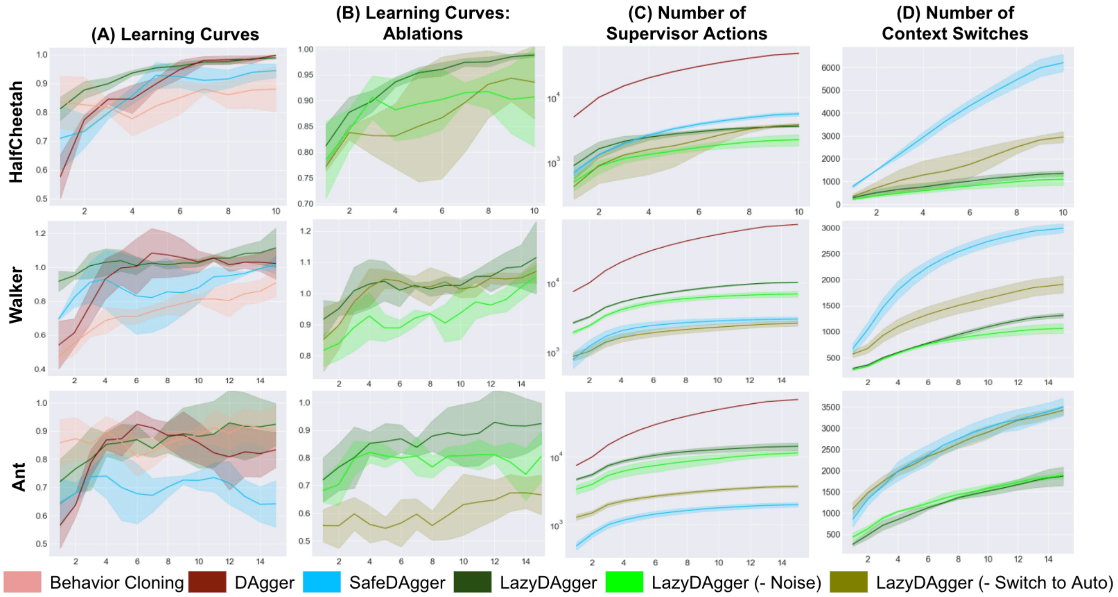


Figure 8.3: **MuJoCo Simulation Results:** We study task performance (A), ablations (B), online supervisor burden (C), and total bidirectional context switches (D) for LazyDAgger and baselines over 3 random seeds. For Columns (A)-(D), the x-axis for all plots shows the number of epochs over the training dataset, while the y-axes indicate normalized reward (A, B), counts of supervisor actions (C, log scale), and context switches (D) with shading for 1 standard deviation. We find that LazyDAgger outperforms all baselines and ablations, indicating that encouraging lengthy, noisy interventions improves performance. Additionally, LazyDAgger uses far fewer context switches than other baselines while requesting far fewer supervisor actions than DAgger.

back to autonomous mode when the action discrepancy falls below a threshold β_R, where $\beta_R < \beta_H$. As illustrated in Figure 8.2, LazyDAgger's asymmetric switching criteria create a hysteresis band, as is often utilized in control theory [227]. Motivated by Eq. (8.3), we adjust β_H to reduce context switches $C(\pi)$ and adjust β_R as a function of β_H to increase intervention length. We hypothesize that redistributing the supervisor actions into fewer but longer sequences in this fashion both reduces burden on the supervisor and improves the quality of the online feedback for the robot. Details on setting these hyperparameter values in practice, the settings used in our experiments, and a hyperparameter sensitivity analysis are provided in the Appendix.

8.5.2 Noise Injection

If the safety classifier is querying for interventions at state s_t, then the robot either does not have much experience in the neighborhood of s_t or has trouble matching the demonstrations at s_t. This motivates exploring novel states near s_t so that the robot can receive maximal feedback on the correct behavior in areas of the state space where it predicts a large action

Algorithm 4 The Student-Teacher Framework

```
 1: Collect $\mathcal{D}, \mathcal{D}_S$ offline with supervisor policy $\pi_H$
 2: $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t))\right]$    ▷ Eq. (8.1)
 3: $f \leftarrow \arg\min_f \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}\cup\mathcal{D}_S} \left[\mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f)\right]$    ▷ Eq. (8.4)
 4: for $i \in \{1, \dots N\}$ do
 5:     Initialize $s_0$, Mode ← Autonomous
 6:     for $t \in \{1, \dots T\}$ do
 7:         $a_t \sim \pi_R(s_t)$
 8:         if Mode = Supervisor or $f(s_t) \geq 0.5$ then
 9:             $a_t^H = \pi_H(s_t)$
10:             $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t^H)\}$
11:             Execute $\tilde{a}_t^H \sim \mathcal{N}(a_t^H, \sigma^2 I)$
12:             if $\mathcal{L}(a_t, a_t^H) < \beta_R$ then
13:                 Mode ← Autonomous
14:             else
15:                 Mode ← Supervisor
16:             end if
17:         else
18:             Execute $a_t$
19:         end if
20:     end for
21:     $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t))\right]$
22:     $f \leftarrow \arg\min_f \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}\cup\mathcal{D}_S} \left[\mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f)\right]$
23: end for
```

discrepancy from the supervisor. Inspired by prior work that has identified noise injection as a useful tool for improving the performance of imitation learning algorithms (e.g. Laskey et al. [204] and Brown et al. [199]), we diversify the set of states visited in supervisor mode by injecting isotropic Gaussian noise into the supervisor's actions, where the variance σ^2 is a scalar hyperparameter (Line 11 in Algorithm 1).

8.6 Experiments

We study whether LazyDAgger can (1) reduce supervisor burden while (2) achieving similar or superior task performance compared to prior algorithms. Implementation details are provided in the supplementary material. In all experiments, $\mathcal{L}$ measures Euclidean distance.

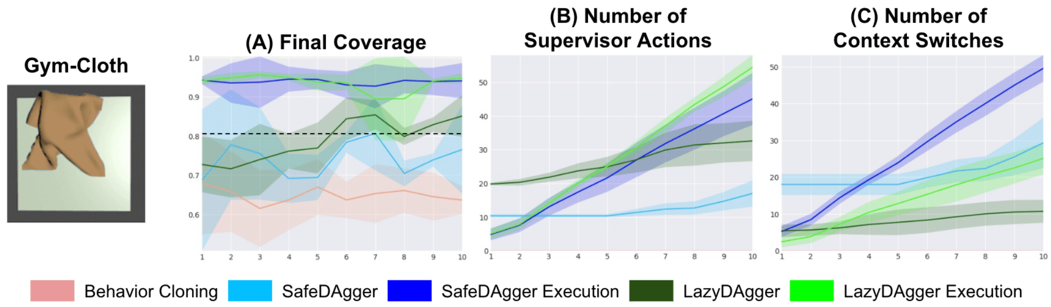


Figure 8.4: **Fabric Smoothing Simulation Results:** We study task performance measured by final fabric coverage (A), total supervisor actions (B), and total context switches (C) for LazyDAgger and baselines in the Gym-Cloth environment from [228]. The horizontal dotted line shows the success threshold for fabric smoothing. LazyDAgger achieves higher final coverage than Behavior Cloning and SafeDAgger with fewer context switches than SafeDAgger but more supervisor actions. At execution time, we again observe that LazyDAgger achieves similar coverage as SafeDAgger but with fewer context switches.

8.6.1 Simulation Experiments: MuJoCo Benchmarks

Environments: We evaluate LazyDAgger and baselines on 3 continuous control environments from MuJoCo [229], a standard simulator for evaluating imitation and reinforcement learning algorithms. In particular, we evaluate on HalfCheetah-v2, Walker2D-v2 and Ant-v2.

Metrics: For LazyDAgger and all baselines, we report learning curves which indicate how quickly they can make task progress in addition to metrics regarding the burden imposed on the supervisor. To study supervisor burden, we report the number of supervisor actions, the number of context switches, and the total supervisor burden (as defined in Eq. (8.2)). Additionally, we define $L^* \geq 0$ to be the latency value such that for all $L > L^*$, LazyDAgger has a lower supervisor burden than SafeDAgger. We report this L^* value, which we refer to as the *cutoff latency*, for all experiments to precisely study the types of domains in which LazyDAgger is most applicable.

Baselines: We compare LazyDAgger to Behavior Cloning [195], DAgger [122], and SafeDAgger [19] in terms of the total supervisor burden and task performance. The Behavior Cloning and DAgger comparisons evaluate the utility of human interventions, while the comparison to SafeDAgger, another interactive IL algorithm, evaluates the impact of soliciting fewer but longer interventions.

Experimental Setup: For all MuJoCo environments, we use a reinforcement learning agent trained with TD3 [63] as an algorithmic supervisor. We begin all LazyDAgger, SafeDAgger, and DAgger experiments by pre-training the robot policy with Behavior Cloning on 4,000 state-action pairs for 5 epochs, and similarly report results for Behavior Cloning after the 5th epoch. To ensure a fair comparison, Behavior Cloning uses additional offline data equal to the average amount of online data seen by LazyDAgger during training. All results are averaged over 3 random seeds.

Results: In Figure 8.3, we study the performance of LazyDAgger and baselines. After every epoch of training, we run the policy for 10 test rollouts *where interventions are not allowed* and report the task reward on these rollouts in Figure 8.3. Results suggest that LazyDAgger is able to match or outperform all baselines in terms of task performance across all simulation environments (Figure 8.3A). Additionally, LazyDAgger requires far fewer context switches compared to SafeDAgger (Figure 8.3D), while requesting a similar number of supervisor actions across domains (Figure 8.3C): we observe a 79%, 56%, and 46% reduction in context switches on the HalfCheetah, Walker2D, and Ant environments respectively. LazyDAgger and SafeDAgger both use an order of magnitude fewer supervisor actions than DAgger. While SafeDAgger requests much fewer supervisor actions than LazyDAgger in the Ant environment, this limited amount of supervision is insufficient to match the task performance of LazyDAgger or any of the baselines, suggesting that SafeDAgger may be terminating interventions prematurely. We study the total supervisor burden of SafeDAgger and LazyDAgger as defined in Equation (8.2) and find that in HalfCheetah, Walker2D, and Ant, the cutoff latencies L^* are 0.0, 4.3, and 7.6 respectively, i.e. LazyDAgger achieves lower supervisor burden in the HalfCheetah domain for any L as well as lower burden in Walker2D and Ant for $L > 4.3$ and $L > 7.6$ respectively. The results suggest that LazyDAgger can reduce total supervisor burden compared to SafeDAgger even for modest latency values, but that SafeDAgger may be a better option for settings with extremely low latency.

Ablations: We study 2 key ablations for LazyDAgger in simulation: (1) returning to autonomous mode with $f(\cdot)$ rather than using the ground truth discrepancy (LazyDAgger (-Switch to Auto) in Figure 8.3), and (2) removal of noise injection (LazyDAgger (-Noise)). LazyDAgger outperforms both ablations on all tasks, with the exception of ablation 1 on Walker2D, which performed similarly well. We also observe that LazyDAgger consistently requests more supervisor actions than either ablation. This aligns with the intuition that both using the ground truth action discrepancy to switch back to autonomous mode and injecting noise result in longer but more useful interventions that improve performance.

Algorithm	Task Successes	Task Progress			Context Switches	Supervisor Actions	Robot Actions	Failure Modes			
		(1)	(2)	(3)				A	B	C	D
Behavior Cloning	0/10	6/10	0/10	0/10	N/A	N/A	119	2	1	7	0
SD-Execution	2/10	6/10	4/10	2/10	53	**34**	108	5	0	0	3
LD-Execution	**8/10**	10/10	10/10	8/10	**21**	43	47	0	0	0	2

Table 8.1: **Physical Fabric Manipulation Experiments:** We evaluate LazyDAgger-Execution and baselines on a physical 3-stage fabric manipulation task and report the success rate and supervisor burden in terms of total supervisor actions and bidirectional context switches (summed across all 10 trials). Task Progress indicates how many trials completed each of the 3 stages: Smoothing, Aligning, and Folding. LazyDAgger-Execution achieves more successes with fewer context switches ($L^* = 0.28$). We observe the following failure modes (Table 8.1): (A) action limit hit (> 15 total actions), (B) fabric is more than 50% out of bounds, (C) incorrect predicted pick point, and (D) the policy failed to request an intervention despite high ground truth action discrepancy.

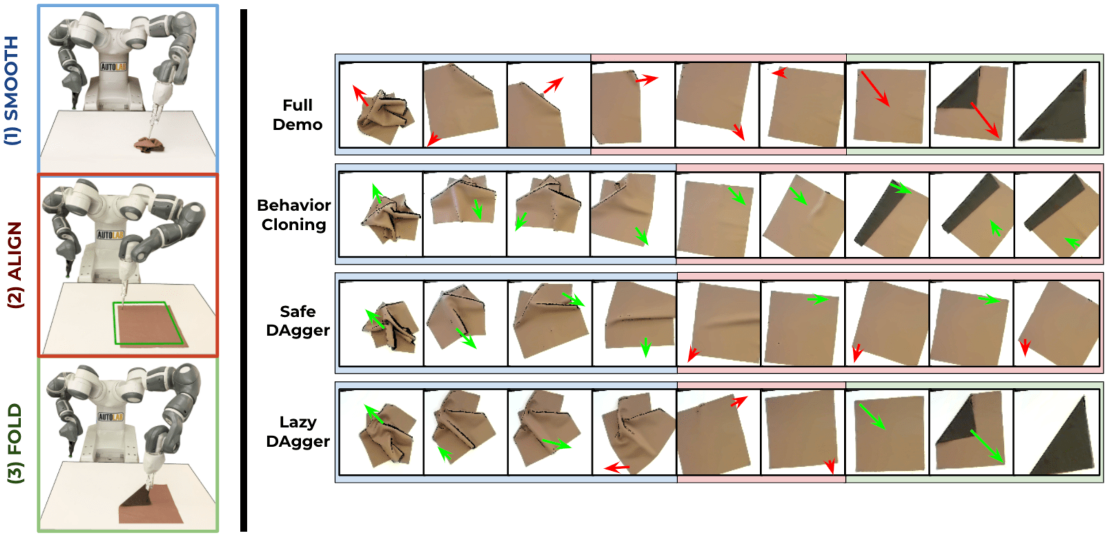


Figure 8.5: **Physical Fabric Manipulation Task:** *Left:* We evaluate on a 3-stage fabric manipulation task consisting of smoothing a crumpled fabric, aligning the fabric so all corners are visible in the observations, and performing a triangular fold. *Right:* Rollouts of the fabric manipulation task, where each frame is a 100 × 100 × 3 overhead image. Human supervisor actions are denoted in red while autonomous robot actions are in green. Rollouts are shaded to indicate task progress: blue for smoothing, red for alignment, and green for folding. SafeDAgger ends human intervention prematurely, resulting in poor task performance and more context switches, while LazyDAgger switches back to robot control only when confident in task completion.

8.6.2 Fabric Smoothing in Simulation

Environment: We evaluate LazyDAgger on the fabric smoothing task from [228] (shown in Figure 8.4) using the simulation environment from [228]. The task requires smoothing an initially crumpled fabric and is challenging due to the infinite-dimensional state space and complex dynamics, motivating learning from human feedback. As in prior work [228], we utilize top-down 100 × 100 × 3 RGB image observations of the workspace and use actions which consist of a 2D pick point and a 2D pull vector. See [228] for further details on the fabric simulator.

Experimental Setup: We train a fabric smoothing policy in simulation using DAgger under supervision from an analytic corner-pulling policy that leverages the simulator's state to identify fabric corners, iterate through them, and pull them towards the corners of the workspace [228]. We transfer the resulting policy for a 16×16 grid of fabric into a new simulation environment with altered fabric dynamics (i.e. lower spring constant, altered fabric colors, and a higher-fidelity 25×25 discretization) and evaluate LazyDAgger and baselines on how rapidly they can adapt the initial policy to the new domain. As in [228], we terminate rollouts when we exceed 10 time steps, 92% coverage, or have moved the fabric more than 20% out of bounds. We evaluate performance based on a coverage metric, which measures the percentage of the background plane that the fabric covers (fully smooth corresponds to a coverage of 100).

Results: We report results for the fabric smoothing simulation experiments in Figure 8.4. Figure 8.4 (A) shows the performance of the SafeDAgger and LazyDAgger policies during learning. To generate this plot we periodically evaluated each policy on *test* rollouts without interventions. Figure 8.4 (B) and (C) show the number of supervisor actions and context switches required during learning; LazyDAgger performs fewer context switches than SafeDAgger but requires more supervisor actions as the interventions are longer. Results suggest that the cutoff latency (as defined in Section 8.6.1) is $L^* = 1.5$ for fabric smoothing. Despite fewer context switches, LazyDAgger achieves comparable performance to SafeDAgger, suggesting that LazyDAgger can learn complex, high-dimensional robotic control policies while reducing the number of hand-offs to a supervisor. We also evaluate LazyDAgger-Execution and SafeDAgger-Execution, in which interventions are allowed but the policy is no longer updated (see Section 8.6.3). We see that in this case, LazyDAgger achieves similar final coverage as SafeDAgger with significantly fewer context switches.

8.6.3 Physical Fabric Manipulation Experiments

Environment: In physical experiments, we evaluate on a multi-stage fabric manipulation task with an ABB YuMi robot and a human supervisor (Figure 8.5). Starting from a crumpled initial fabric state, the task consists of 3 stages: (1) fully smooth the fabric, (2) align the fabric corners with a tight crop of the workspace, and (3) fold the fabric into a triangular fold. Stage (2) in particular requires high precision, motivating human interventions. As in the fabric simulation experiments, we use top-down $100 \times 100 \times 3$ RGB image observations of the workspace and have 4D actions consisting of a pick point and pull vector. The actions are converted to workspace coordinates with a standard calibration procedure and analytically mapped to the nearest point on the fabric. Human supervisor actions are provided through a point-and-click interface for specifying pick-and-place actions. See the supplement for further details.

Experimental Setup: Here we study how interventions can be leveraged to improve the final task performance even at *execution time*, in which policies are no longer being updated. We collect 20 offline task demonstrations and train an initial policy with behavior cloning. To prevent overfitting to a small amount of real data, we use standard data augmentation techniques such as rotating, scaling, changing brightness, and adding noise to create 10 times as many training examples. We then evaluate the behavior cloning agent (Behavior Cloning) and agents which use the SafeDAgger and LazyDAgger intervention criteria but do not update the policy with new experience or inject noise (SafeDAgger-Execution and LazyDAgger-Execution respectively). We terminate rollouts if the fabric has successfully reached the goal state of the final stage (i.e. forms a perfect or near-perfect dark brown right triangle as in Hoque et al. [230]; see Figure 8.5), more than 50% of the fabric mask is out of view in the current observation, the predicted pick point misses the fabric mask by approximately 50% of the plane or more, or 15 total actions have been executed (either autonomous or supervisor).

Results: We perform 10 physical trials of each technique. In Table 8.1, we report both the overall task success rate and success rates for each of the three stages of the task: (1) Smoothing, (2) Alignment, and (3) Folding. We also report the total number of context switches, supervisor actions, and autonomous robot actions summed across all 10 trials for each algorithm (Behavior Cloning, SafeDAgger-Execution, LazyDAgger-Execution). In Figure 8.5 we provide representative rollouts for each algorithm. Results suggest that Behavior Cloning is insufficient for successfully completing the alignment stage with the required level of precision. SafeDAgger-Execution does not improve the task success rate significantly due to its inability to collect interventions long enough to navigate bottleneck regions in the task (Figure 8.5). LazyDAgger-Execution, however, achieves a much higher success rate than SafeDAgger-Execution and Behavior Cloning with far fewer context switches than SafeDAgger-Execution: LazyDAgger-Execution requests 2.1 context switches on average per trial (i.e. 1.05 interventions) as opposed to 5.3 switches (i.e. 2.65 interventions). LazyDAgger-Execution trials also make far more task progress than the baselines, as all 10 trials reach the folding stage. LazyDAgger-Execution does request more supervisor actions than SafeDAgger-Execution, as in the simulation environments. LazyDAgger-Execution also requests more supervisor actions relative to the total amount of actions due to the more conservative switching criteria and the fact that successful episodes are shorter than unsuccessful episodes on average. Nevertheless, results suggest that for this task, LazyDAgger-Execution reduces supervisor burden for any $L > L^* = 0.28$, a very low cutoff latency that includes all settings in which a context switch is at least as time-consuming as an individual action (i.e. $L \geq 1$).

In experiments, we find that SafeDAgger-Execution's short interventions lead to many instances of Failure Mode A (see Table 8.1), as the policy is making task progress, but not quickly enough to perform the task. We observe that Failure Mode C is often due to the fabric reaching a highly irregular configuration that is not within the training data distribution, making it difficult for the robot policy to make progress. We find that SafeDAgger and LazyDAgger experience Failure Mode D at a similar rate as they use the same criteria to solicit interventions (but different termination criteria). However, we find that all of LazyDAgger's failures are due to Failure Mode D, while SafeDAgger also fails in Mode A due to premature termination of interventions.

8.7 Discussion and Future Work

We propose context switching between robot and human control as a metric for supervisor burden in interactive imitation learning and present LazyDAgger, an algorithm which can be used to efficiently learn tasks while reducing this switching. We evaluate LazyDAgger on 3 continuous control benchmark environments in MuJoCo, a fabric smoothing environment in simulation, and a fabric manipulation task with an ABB YuMi robot and find that

LazyDAgger is able to improve task performance while reducing context switching between the learner and robot by up to 79% over SafeDAgger. In subsequent work, we investigate more intervention criteria and apply robot-gated interventions to controlling a fleet of robots, where context switching can negatively impact task throughput.

Part IV

Conclusion

Chapter 9

Concluding Remarks

This thesis presents a set of safe reinforcement learning algorithms that use data from demonstrations and online experience to construct regions of the state space that are used to avoid unsafe behaviors during online exploration. The first class of algorithms maintains distance to regions where prior policy iterates are confident in task success. The second class of algorithms maintains distance from likely constraint violations by learning their likelihood from prior experience and demonstrations. The third class uses policy uncertainty to solicit human interventions. In all classes, a plan to recover to safety exists, by maintaining reachability to a prior policy's safe set, querying a recovery policy, or calling a human supervisor. These algorithms may be naturally adaptable to the multi-agent setting, where a small set of human supervisors manages a much larger fleet of robots by sharing a risk budget across robots. In future work, I hope to further explore this generalization and formalize this setting, which is increasingly relevant as fleets of robots are deployed in the real world for applications such as delivery, manufacturing, and kitting. In this section, I will elaborate on lessons learned and future ideas.

9.1 Lessons Learned

Over the course of my PhD, I have had the opportunity to work on dozens of research projects on a diverse set of topics with well over fifty collaborators. In this subsection I will discuss a few principles I have adopted over the years, which I hope might be helpful to a future reader someday.

9.1.1 Power of Collaboration

As alluded to throughout this thesis, collaborations were one of the most rewarding and productive aspects of my PhD. My most successful collaborations were ones that involve collaborators with complementary backgrounds, where each person can bring a different skill set or perspective to the project. This class of collaboration is especially instructive because I often learn how different communities can view the same problems. Additionally, these collaborations provide the opportunity to quickly learn new skills from experts, which become lifelong tools that I can use thereafter.

9.1.2 High Level Focus

A second lesson I learned throughout the course of my PhD is to not lose track of the high level objective. I found it easy to get lost in the details and sometimes the beauty of methods, which can take research in a direction orthogonal to the ultimate objective. As an analogy, this is like getting lost in the details of a proof or a paper and forgetting its high level structure. I found it very useful to have constant awareness of the greater context of anything I worked on, and I tried to instill this in the undergraduates I worked with. This is a skill I work on continuously, and it requires balancing between detail orientation (which is also important) and high level understanding.

9.1.3 Starting Simple

I also found that it was helpful to start off with ideas that seem relatively “obvious.” In my experience, even obvious ideas usually end up being more nuanced than expected. These unexpected nuances are exacerbated by the complexity of the idea or problem, which can lead to lack of progress or confusion, because it becomes more difficult to isolate the cause of phenomena. So, I am constantly running sanity checks and obvious experiments, which often lead to discoveries when something unexpected inevitably occurs.

9.1.4 The Value of Simulation for Real World Learning

During my study of safe reinforcement learning, I found that safety is critical for many real world reinforcement learning tasks. However, prototyping algorithms in real systems is challenging, due to engineering overhead and the time-intensive and sequential nature of most physical experimental setups. So, I found that prototyping algorithms in simulations to be very important, and while this thesis targets methods that would be applicable to real world reinforcement learning, simulation is still a critical technology their advancement. However, I do wish I spent more of my PhD trying these algorithms on a variety of different real systems and tasks. This was partially limited by the outbreak of COVID 19, but I hope to run more real world experiments in the future.

9.2 Future Directions in Parallel Safe Reinforcement Learning

The algorithms in this work suggest that human supervision is critical for real world robot learning, from providing resets to corrective feedback. However, supervising each robot individually with a human is expensive and does not scale well. However, when running robot experiments, direct supervision is not always required. This is a limitation I found when I was running robot experiments: I had to be nearby constantly in case of hard failures, but I was mostly idle. I always wondered if it may be possible for a single person to effectively supervise a set of several robots learning in parallel.

The algorithms discussed in this work suggest a natural extension to a setting that contains m different robots learning in parallel with $n < m$ humans that are assigned to supervise them. The robots may have different morphologies and may be performing tasks, but are still supervised by the same set of human supervisors. In practice, this might mean a warehouse containing m robots or a fleet of m delivery drones, all semi-autonomously exploring or executing their policies in the real world. The goal is to design an intelligent supervisor allocation strategy that assigns humans to perform interventions to specific robots in order to maximize a collective performance metric. An intervention could consist of corrective feedback for imitation learning, recovery actions to take the robot away from danger, or resets to address the consequences of unsafe behaviors. An intelligent supervisor allocation strategy will consider the consequences of each of these type of interventions and the state of all of the robots before assigning a human to perform an intervention.

There are two settings worth discussing here: *training time* and *execution time*. During training time, the robot policies are all learning in the real world together and the goal is to maximize the collective learning progress of the robots. During execution time, the robot policies, which may not be learning-based at all, the goal is to maximize the collective throughput of the robots.

An intelligent allocation strategy will consider the real world cost of failures when assigning humans. For example, a constraint violation may be worth avoiding at all costs, because it can result in long human reset times and a high real world cost. So, robots that are in danger of failing catastrophically should be prioritized over robots that are unsure but are safe. An allocation strategy should also consider when robots should be paused: if the strategy is unable to supervise the robots effectively, it should be able to report that it is overloaded. In future work, I hope to explore different methods for human supervisor allocation during real world robot learning in order to address these concerns.

Bibliography

[1] Ronald A Howard and James E Matheson. “Risk-sensitive Markov decision processes”. In: *Management science* 18.7 (1972), pp. 356–369.

[2] J. Garcia and F. Fernández. “A Comprehensive Survey on Safe Reinforcement Learning”. In: *Journal of Machine Learning Research*. 2015.

[3] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. “Reward Constrained Policy Optimization”. In: *International Conference on Learning Representations (ICLR)*. 2019.

[4] Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. “Conservative Safety Critics for Exploration”. In: *arXiv preprint arXiv:2010.14497* (2020).

[5] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. “Constrained policy optimization”. In: *Journal of Machine Learning Research (JMLR)*. 2017.

[6] Fritz Wysotzki Peterr Geibel. “Risk-Sensitive Reinforcement Learning Applied to Control under Constraints”. In: *Journal of Artificial Intelligence Rersearch*. Vol. 24. 2005.

[7] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. “Learning to be Safe: Deep RL with a Safety Critic”. In: *arXiv preprint arXiv:2010.14603* (2020).

[8] Brijen Thananjeyan, Ashwin Balakrishna, Suraj Nair, Michael Luo, Krishnan Srinivasan, Minho Hwang, Joseph E. Gonzalez, Julian Ibarz, Chelsea Finn, and Ken Goldberg. “Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones”. In: *Robotics and Automation Letters (RAL)*. IEEE. 2020.

[9] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. “Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning”. In: *International Conference on Learning Representations* (2018).

[10] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. “Safety Augmented Value Estimation from Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks”. In: *IEEE Robotics and Automation Letters (RA-L)*. 2020.

[11] Jaime F. Fisac, Anayo K. Akametalu, Melanie Nicole Zeilinger, Shahab Kaynama, Jeremy H. Gillula, and Claire J. Tomlin. "A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems". In: *CoRR* abs/1705.01292 (2017). arXiv: `1705.01292`.

[12] Somil Bansal, Mo Chen, Sylvia Herbert, and Claire J. Tomlin. "Hamilton-Jacobi Reachability: A Brief Overview and Recent Advances". In: *Conference on Decision and Control (CDC)*. 2017.

[13] U. Rosolia, X. Zhang, and F. Borrelli. "Robust Learning Model Predictive Control for Iterative Tasks: Learning from Experience". In: *Annual Conference on Decision and Control (CDC)*. 2017.

[14] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Joseph E. Gonzalez, Aaron Ames, and Ken 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.

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

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

[17] Ryan Hoque, Ashwin Balakrishna, Carl Putterman, Michael Luo, Daniel S. Brown, Daniel Seita, Brijen Thananjeyan, Ellen Novoseller, and Ken Goldberg. "LazyDAgger: Reducing Context Switching in Interactive Imitation Learning". In: *IEEE Conference on Automation Science and Engineering (CASE)*. 2021.

[18] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. "HG-DAgger: Interactive Imitation Learning with Human Experts". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[19] Jiakai Zhang and Kyunghyun Cho. "Query-Efficient Imitation Learning for End-to-End Autonomous Driving". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2017.

[20] Kunal Menda, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. "EnsembleDAgger: A Bayesian Approach to Safe Imitation Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019.

[21] Ajay Mandlekar, Danfei Xu, Roberto Martín-Martín, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. *Human-in-the-Loop Imitation Learning using Remote Teleoperation*. 2020. arXiv: `2012.06733 [cs.RO]`.

[22] Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S Brown, and Ken Goldberg. "ThriftyDAgger: Budget-aware novelty and risk gating for interactive imitation learning". In: *arXiv preprint arXiv:2109.08273* (2021).

[23] Michael Laskey, Sam Staszak, Wesley Yu-Shu Hsieh, Jeffrey Mahler, Florian T Pokorny, Anca D Dragan, and Ken Goldberg. “Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[24] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E. Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. “Safety Augmented Value Estimation From Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3612–3619.

[25] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. “Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models”. In: *Neural Information Processing Systems (NeurIPS)* (2018).

[26] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. “Deep Dynamics Models for Learning Dexterous Manipulation”. In: *Conference on Robot Learning (CoRL)* (2019).

[27] Ashwin Balakrishna, Brijen Thananjeyan, Jonathan Lee, Arsh Zahed, Felix Li, Joseph E. Gonzalez, and Ken Goldberg. “On-Policy Robot Imitation Learning from a Converging Supervisor”. In: *Conference on Robot Learning (CoRL)*. 2019.

[28] U. Rosolia, A. Carvalho, and F. Borrelli. “Autonomous Racing using Learning Model Predictive Control”. In: *Proceedings 2017 IFAC World Congress*. 2017.

[29] Anil Aswani, Humberto Gonzalez, Shankar Sastry, and Claire Tomlin. “Provably Safe and Robust Learning-Based Model Predictive Control”. In: *Automatica* 49 (2011).

[30] Jus Kocijan, Roderick Murray-Smith, C.E. Rasmussen, and A. Girard. “Gaussian process model based predictive control”. In: *American Control Conference (ACC)*. 2004.

[31] Torsten Koller, Felix Berkenkamp, Matteo Turchetta, and Andreas Krause. “Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning”. In: 2018.

[32] Lukas Hewing, Alexander Liniger, and Melanie Zeilinger. “Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars”. In: *European Controls Conference (ECC)*. 2018.

[33] Enrico Terzi, Lorenzo Fagiano, Marcello Farina, and Riccardo Scattolini. “Learning-based predictive control for linear systems: A unitary approach”. In: *Automatica* 108 (2019).

[34] Lukas Hewing, Juraj Kabzan, and Melanie N Zeilinger. “Cautious model predictive control using Gaussian process regression”. In: *IEEE Transactions on Control Systems Technology* (2019).

[35] Cheng-Yu Kuo, Andreas Schaarschmidt, Yunduan Cui, Tamim Asfour, and Takamitsu Matsubara. “Uncertainty-aware contact-safe model-based reinforcement learning”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3918–3925.

[36] Marko Bacic, Mark Cannon, Young Il Lee, and Basil Kouvaritakis. “General interpolation in MPC and its advantages”. In: *IEEE Transactions on Automatic Control* 48 (2003).

[37] Florian D Brunner, Mircea Lazar, and Frank Allgöwer. “Stabilizing linear model predictive control: On the enlargement of the terminal set”. In: *2013 European Control Conference (ECC)*. 2013.

[38] Kim P Wabersich and Melanie N Zeilinger. “Linear model predictive safety certification for learning-based control”. In: *2018 IEEE Conference on Decision and Control (CDC)*. 2018.

[39] Franco Blanchini and Felice Andrea Pellegrino. “Relatively optimal control and its linear implementation”. In: *IEEE Transactions on Automatic Control* 48 (2003).

[40] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. “Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control”. In: *International Conference on Machine Learning (ICML)*. 2019.

[41] Lukas Hewing, Kim P Wabersich, Marcel Menner, and Melanie N Zeilinger. “Learning-Based Model Predictive Control: Toward Safe Learning in Control”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 3 (2019), pp. 269–296.

[42] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. “Reverse Curriculum Generation for Reinforcement Learning”. In: *Conference on Robot Learning (CoRL)*. Vol. abs/1707.05300. 2017. arXiv: `1707.05300`.

[43] Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alex Peysakhovich, Kyunghyun Cho, and Joan Bruna. “Backplay: ”Man muss immer umkehren””. In: *CoRR* abs/1807.06919 (2018). arXiv: `1807.06919`.

[44] Sanmit Narvekar and Peter Stone. “Learning Curriculum Policies for Reinforcement Learning”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019.

[45] Boris Ivanovic, James Harrison, Apoorva Sharma, Mo Chen, and Marco Pavone. “BaRC: Backward Reachability Curriculum for Robotic Reinforcement Learning”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[46] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. “Visual Reinforcement Learning with Imagined Goals”. In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[47] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. “Universal Value Function Approximators”. In: *International Conference on Machine Learning (ICML)*. 2015.

[48] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5048–5058.

[49] Jur van den Berg, Pieter Abbeel, and Kenneth Y. Goldberg. "LQG-MP: Optimized path planning for robots with motion uncertainty and imperfect state information." In: *I. J. Robotics Res.* 30.7 (2011), pp. 895–913.

[50] Alex Pui-wai Lee, Sachin Patil, John Schulman, Zoe McCarthy, Jur van den Berg, Ken Goldberg, and Pieter Abbeel. "Gaussian Belief Space Planning for Imprecise Articulated Robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[51] Hanna Kurniawati, Yanzhu Du, David Hsu, and Wee Sun Lee. "Motion planning under uncertainty for robotic tasks with long time horizons". In: *International Journal of Robotics Research (IJRR)* 30.3 (2011), pp. 308–323.

[52] Charles Richter and Nicholas Roy. "Safe Visual Navigation via Deep Learning and Novelty Detection". In: *Robotics Science and Systems (RSS)* (2013).

[53] Mikael Jorda, Elena Galbally Herrero, and Oussama Khatib. "Contact-Driven Posture Behavior for Safe and Interactive Robot Operation". In: *International Conference on Robotics and Automation (ICRA)* (2019).

[54] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[55] Anusha Nagabandi, Gregory Kahn, Ronald Fearing, and Sergey Levine. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[56] MP. Deisenroth and CE. Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *International Conference on Machine Learning (ICML)*. 2011.

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

[58] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4019–4026.

[59] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. "Concrete Problems in AI Safety". In: *arXiv preprint arXiv:1606.06565* (June 2016).

[60] D. Seita, S. Krishnan, R. Fox, S. McKinley, J. Canny, and K. Goldberg. "Fast and Reliable Autonomous Surgical Debridement with Cable-Driven Robots Using a Two-Phase Calibration Procedure". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[61] Arkadi Nemirovski. "On safe tractable approximations of chance constraints". In: *European Journal of Operational Research* (2012).

[62] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)* (2018).

[63] Scott Fujimoto, Herke van Hoof, and David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". In: *International Conference on Machine Learning (ICML)*. 2018.

[64] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. "Deep q-learning from demonstrations". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[65] Matej Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A. Riedmiller. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *CoRR* abs/1707.08817 (2017).

[66] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. "Overcoming Exploration in Reinforcement Learning with Demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[67] Stephen Tu and Benjamin Recht. "The Gap Between Model-Based and Model-Free Methods on the Linear Quadratic Regulator: An Asymptotic Viewpoint". In: *CoRR* abs/1812.03565 (2018).

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

[69] Daniel Brown, Wonjoon Goo, Prabhat Nagarajan, and Scott Niekum. "Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 783–792.

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

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

[72] Javier García and Fernando Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research (JMLR)* 16.1 (2015), pp. 1437–1480.

[73] Z. Li, U. Kalabić, and T. Chu. "Safe Reinforcement Learning: Learning with Supervision Using a Constraint-Admissible Set". In: *2018 Annual American Control Conference (ACC)*. 2018.

[74] Teodor Mihai Moldovan and Pieter Abbeel. "Safe exploration in Markov decision processes". In: *arXiv preprint arXiv:1205.4810* (2012).

[75] Felix Berkenkamp, Matteo Turchetta, Angela P. Schoellig, and Andreas Krause. "Safe Model-based Reinforcement Learning with Stability Guarantees". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[76] Teodor M. Moldovan and Pieter Abbeel. "Risk Aversion in Markov Decision Processes via Near Optimal Chernoff Bounds". In: *Neural Information Processing Systems (NeurIPS)*. 2012.

[77] Takayuki Osogami. "Robustness and risk-sensitivity in Markov decision processes". In: *neurips*. 2012.

[78] Peter Kazanzides, Zihan Chen, Anton Deguet, Gregory S. Fischer, Russell H. Taylor, and Simon P. DiMaio. "An Open-Source Research Kit for the da Vinci Surgical System". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[79] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research". In: *CoRR* abs/1802.09464 (2018). arXiv: `1802.09464`.

[80] Jur Van Den Berg, Stephen Miller, Daniel Duckworth, Humphrey Hu, Andrew Wan, Xiao-Yu Fu, Ken Goldberg, and Pieter Abbeel. "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010.

[81] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control". In: *arXiv preprint arXiv:1812.00568* (2018).

[82] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. "Learning Latent Dynamics for Planning from Pixels". In: *International Conference on Machine Learning (ICML)* (2018).

[83] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Kumar Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. "Visuospatial foresight for multi-step, multi-task fabric manipulation". In: *Robotics: Science and Systems (RSS)* (2020).

[84] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. “DeepMPC: Learning deep latent features for model predictive control.” In: *Robotics: Science and Systems*. Rome, Italy. 2015.

[85] Suraj Nair and Chelsea Finn. “Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation”. In: *International Conference on Learning Representations (ICLR)* (2019).

[86] Suraj Nair, Silvio Savarese, and Chelsea Finn. “Goal-Aware Prediction: Learning to Model What Matters”. In: *International Conference on Machine Learning (ICML)* (2020), pp. 7207–7219.

[87] Karl Pertsch, Oleh Rybkin, Frederik Ebert, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. “Long-horizon visual planning with goal-conditioned hierarchical predictors”. In: *Neural Information Processing Systems (NeurIPS)* (2020).

[88] Stephen Tian, Suraj Nair, Frederik Ebert, Sudeep Dasari, Benhamin Eysenbach, Chelsea Finn, and Sergey Levine. “Model-Based Visual Planning with Self-Supervised Functional Distances”. In: *International Conference on Learning Representations (ICLR)* (2021).

[89] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. “Soft Actor-Critic Algorithms and Applications”. In: *CoRR* (2018).

[90] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research (JMLR)* (2016).

[91] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. “Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation”. In: *Conference on Robot Learning (CoRL)* (2018).

[92] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. “Skew-fit: State-covering self-supervised reinforcement learning”. In: *International Conference on Machine Learning (ICML)* (2020).

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

[94] Jeffrey Ichnowski, Yahav Avigal, Vishal Satish, and Ken Goldberg. “Deep learning can accelerate grasp-optimized motion planning”. In: *Science Robotics* 5.48 (2020).

[95] Zhaoxuan Zhu, Nicola Pivaroa, Shobhit Gupta, Abhishek Gupta, and Marcello Canova. In: (2021). arXiv: `2105.11640 [cs.LG]`.

[96] Martina Lippi, Petra Poklukar, Michael C Welle, Anastasiia Varava, Hang Yin, Alessandro Marino, and Danica Kragic. "Latent Space Roadmap for Visual Action Planning of Deformable and Rigid Object Manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2020.

[97] Andrei A Rusu, Matej Večerík, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. "Sim-to-real robot learning from pixels with progressive nets". In: *Conference on Robot Learning*. PMLR. 2017, pp. 262–270.

[98] Gerrit Schoettler, Ashvin Nair, Jianlan Luo, Shikhar Bahl, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. "Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020).

[99] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. "End-to-end robotic reinforcement learning without reward engineering". In: *Robotics: Science and Systems (RSS)* (2019).

[100] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew Johnson, and Sergey Levine. "Solar: Deep structured representations for model-based reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 7444–7453.

[101] Greg Kahn, Adam Villaflor, Vitchyr Pong, Pieter Abbeel, and Sergey Levine. "Uncertainty-Aware Reinforcement Learning for Collision Avoidance". In: *CoRR* (2017).

[102] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. "Universal Planning Networks". In: *International Conference on Machine Learning (ICML)* (Apr. 2018).

[103] B. Ichter and M. Pavone. "Robot Motion Planning in Learned Latent Spaces". In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2407–2414.

[104] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *International Conference on Learning Representations (ICLR)* (2017).

[105] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. "Reinforcement Learning with Augmented Data". In: (2020). arXiv:2004.14990.

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

[107] K. Chua, R. Calandra, Rowan McAllister, and S. Levine. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[108] Jesse Zhang, Brian Cheung, Chelsea Finn, Sergey Levine, and Dinesh Jayaraman. "Cautious adaptation for reinforcement learning in safety-critical settings". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 11055–11065.

[109] Ashvin Nair, Abhishek Gupta, Murtaza Dalal, and Sergey Levine. *AWAC: Accelerating Online Reinforcement Learning with Offline Datasets*. 2021. arXiv: `2006.09359 [cs.LG]`.

[110] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. *dm-control: Software and Tasks for Continuous Control*. 2020. arXiv: `2006.12983 [cs.RO]`.

[111] Peter Kazanzides, Zihan Chen, Anton Deguet, Gregory S Fischer, Russell H Taylor, and Simon P DiMaio. "An open-source research kit for the da Vinci® Surgical System". In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 6434–6439.

[112] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. "One-Shot Visual Imitation Learning via Meta-Learning". In: *Conference on Robot Learning (CoRL)* (2017).

[113] Yuxuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. "Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[114] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[115] Yixin Gao, S. Swaroop Vedula, Carol E. Reiley, Narges Ahmidi, Balakrishnan Varadarajan, Henry C. Lin, Lingling Tao, Luca Zappella, Benjamín Béjar, David D. Yuh, Chi Chiung Grace Chen, René Vidal, Sanjeev Khudanpur, and Gregory D. Hager. "JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) : A Surgical Activity Dataset for Human Motion Modeling". In: *MICCAI Workshop*. 2014.

[116] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. "PLATO: Policy learning using adaptive trajectory optimization". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 3342–3349.

[117] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, and Byron Boots. "Agile off-road autonomous driving using end-to-end deep imitation learning". In: *arXiv preprint arXiv:1709.07174* (2017).

[118] Brijen Thananjeyan, Ashwin Balakrishna, Ugo Rosolia, Felix Li, Rowan McAllister, Joseph E Gonzalez, Sergey Levine, Francesco Borrelli, and Ken Goldberg. "Extending Deep Model Predictive Control with Safety Augmented Value Estimation from Demonstrations". In: *CoRR* (2019).

[119] Thomas Anthony, Zheng Tian, and David Barber. "Thinking fast and slow with deep learning and tree search". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[120] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of go without human knowledge". In: *Nature* (2017).

[121] Wen Sun, Geoffrey J Gordon, Byron Boots, and J Bagnell. "Dual policy iteration". In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[122] Stephane Ross, Geoffrey J Gordon, and J Andrew Bagnell. "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011.

[123] Wen Sun, Arun Venkatraman, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. "Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction". In: *International Conference on Machine Learning (ICML)*. 2017.

[124] Jonathan Lee, Michael Laskey, Ajay Kumar Tanwani, Anil Aswani, and Kenneth Y. Goldberg. "A Dynamic Regret Analysis and Adaptive Regularization Algorithm for On-Policy Robot Imitation Learning". In: *WAFR* (2019).

[125] Ching-An Cheng and Byron Boots. "Convergence of Value Aggregation for Imitation Learning". In: *AISTATS* (2018).

[126] J. Andrew (Drew) Bagnell. *An Invitation to Imitation*. Tech. rep. CMU-RI-TR-15-08. Pittsburgh, PA: Carnegie Mellon University, 2015.

[127] Dean A Pomerleau. *Alvinn: An autonomous land vehicle in a neural network*. Tech. rep. Carnegie-Mellon University, 1989.

[128] Michael Laskey, Caleb Chuck, Jonathan Lee, Jeffrey Mahler, Sanjay Krishnan, Kevin Jamieson, Anca Dragan, and Ken Goldberg. "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[129] Ching-An Cheng, Jonathan Lee, Ken Goldberg, and Byron Boots. "Online Learning with Continuous Variations: Dynamic Regret and Reductions". In: *CoRR* (2019).

[130] Alexis Jacq, Matthieu Geist, Ana Paiva, and Olivier Pietquin. "Learning from a Learner". In: *International Conference on Machine Learning (ICML)*. 2019.

[131] Elad Hazan. "Introduction to online convex optimization". In: *Foundations and Trends in Optimization* 2.3-4 (2016), pp. 157–325.

[132] Martin Zinkevich. "Online convex programming and generalized infinitesimal gradient ascent". In: *International Conference on Machine Learning (ICML)*. 2003.

[133] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International Conference on Machine Learning (ICML)*. 2015.

[134] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. "Model-Ensemble Trust-Region Policy Optimization". In: *International Conference on Learning Representations (ICLR)*. 2018.

[135] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous Control with Deep Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. Vol. abs/1509.02971. 2016. arXiv: `1509.02971`.

[136] J. H. Gillula and C. J. Tomlin. "Guaranteed Safe Online Learning via Reachability: tracking a ground target using a quadrotor". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2012.

[137] Shuo Li and Osbert Bastani. "Robust Model Predictive Shielding for Safe Reinforcement Learning with Stochastic Dynamics". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[138] Alex Ray, Joshua Achiam, and Dario Amodei. "Benchmarking Safe Exploration in Deep Reinforcement Learning". In: *NeurIPS Deep Reinforcement Learning Workshop*. 2019.

[139] M. Heger. "Consideration of risk in reinforcement learning". In: *Machine Learning Proceedings*. 1994.

[140] Yun Shen, Michael J. Tobia, Tobias Sommer, and Klaus Obermayer. "Risk-sensitive Reinforcement Learning". In: *Neural Computation*. Vol. 26. 2014.

[141] A. Tamar, Y. Glassner, and S. Mannor. "Policy Gradients Beyond Expectations: Conditional value-at-risk". In: *CoRR*. 2014.

[142] Yichuan Charlie Tang, Jian Zhang, and Ruslan Salakhutdinov. "Worst Cases Policy Gradients". In: *Conference on Robot Learning (CoRL)* (2019).

[143] Jaime F. Fisac, Neil F. Lugovoy, Vicenç Rubies-Royo, Shromona Ghosh, and Claire J. Tomlin. "Bridging Hamilton-Jacobi Safety Analysis and Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[144] Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. "Safe Exploration in Finite Markov Decision Processes with Gaussian Processes". In: *Neural Information Processing Systems (NeurIPS)*. 2016.

[145] Krishnan Srinivasan, Benjamin Eysenbach, Sehoon Ha, Jie Tan, and Chelsea Finn. "Learning to be Safe: Deep RL with a Safety Critic". In: *arXiv preprint arXiv:2010.14603* (2020).

[146] Y. Chow, O. Nachum, E. Duéñez-Guzmán, and M. Ghavamzadeh. "A Lyapunov-based Approach to Safe Reinforcement Learning". In: *NeurIPS*. 2018.

[147] Y. Chow, O. Nachum, A. Faust, M. Ghavamzadeh, and E. Duéñez-Guzmán. "Lyapunov-based Safe Policy Optimization for Continuous Control". In: *ICML Workshop RL4RealLife*. 2019.

[148] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. "Safe Reinforcement Learning via Shielding". In: 2018.

[149] Weiqiao Han, Sergey Levine, and Pieter Abbeel. "Learning Compound Multi-Step Controllers under Unknown Dynamics". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015).

[150] Eitan Altman. *Constrained Markov Decision Processes.* 1999, p. 260.

[151] Minho Hwang, Brijen Thananjeyan, Samuel Paradis, Daniel Seita, Jeffrey Ichnowski, Danyal Fer, Thomas Low, and Ken Goldberg. "Efficiently Calibrating Cable-Driven Surgical Robots With RGBD Sensing, Temporal Windowing, and Linear and Recurrent Neural Network Compensation". In: *Robotics and Automation Letters (RAL)* (2020).

[152] Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. "Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020).

[153] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. "How to Train Your Robot with Deep Reinforcement Learning: Lessons we Have Learned". In: *International Journal of Robotics Research (IJRR)* (2021).

[154] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. *Offline Meta-Reinforcement Learning with Advantage Weighting.* 2020. arXiv: `2008.06043 [cs.LG]`.

[155] Ron Dorfman, Idan Shenfeld, and Aviv Tamar. *Offline Meta Learning of Exploration.* 2021. arXiv: `2008.02598 [cs.LG]`.

[156] Yinlam Chow, Mohammad Ghavamzadeh, Lucas Janson, and Marco Pavone. "Risk-Constrained Reinforcement Learning with Percentile Risk Criteria". In: *Journal of Machine Learning Research* 1 (2015), pp. 1–49.

[157] Albert Wilcox*, Ashwin Balakrishna*, Brijen Thananjeyan, Joseph E. Gonzalez, and Ken Goldberg. "LS3: Latent Space Safe Sets for Long-Horizon Visuomotor Control of Iterative Tasks". In: *Conference on Robot Learning (CoRL).* 2021.

[158] Felix Berkenkamp and Angela P. Schoellig. "Safe and Robust Learning Control with Gaussian Processes". In: *European Control Conference* (2015).

[159] Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. "Safe controller optimization for quadrotors with Gaussian processes". In: *IEEE International Conference on Robotics and Automation (ICRA).* 2016.

[160] Albert Wu and Jonathan P. How. "Guaranteed Infinite Horizon Avoidance of Unpredictable, Dynamically Constrained Obstacles". In: *Autonomous Robots* (2012).

[161] Meeko Oishi Ian M. Mitchell Mo Chen. "Ensuring Safety of Nonlinear Sampled Data Systems through Reachability". In: *International Federation of Automatic Control* 45 (9 2012), pp. 108–114.

[162] Kostas Margellos and John Lygeros. "Hamilton-Jacobi Formulation for Reach-Avoid Differential Games". In: *Transactions on Automatic Control* 56 (8 2011).

[163] Mo Chen, Qie Hu, Casey Mackin, Jaime Fisac, and Claire Tomlin. "Safe Platooning of Unmanned Aerial Vehicles via Reachability". In: *Conference on Decision and Control* (2015).

[164] Susmit Jha, Vasumathi Raman, Dorsa Sadigh, and Sanjit A. Seshia. "Safe Autonomy Under Perception Uncertainty Using Chance-Constrained Temporal Logic". In: *Transactions on Automated Reasoning* (2017).

[165] Jurgen Schmidhuber. "Evolutionary Principles in Self-Referential Learning. On Learning now to Learn: The Meta-Meta-Meta...-Hook". Diploma Thesis. Technische Universitat Munchen, Germany, 1987.

[166] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule.* Citeseer.

[167] Devang K Naik and Richard J Mammone. "Meta-neural networks that learn by learning". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks.* Vol. 1. IEEE. 1992, pp. 437–442.

[168] Sebastian Thrun and Lorien Pratt. "Learning to learn: Introduction and overview". In: *Learning to learn.* Springer, 1998, pp. 3–17.

[169] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. "Learning To Learn Using Gradient Descent". In: *IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001.* Springer, 2001, pp. 87–94.

[170] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning.* 2016. arXiv: `1611.02779 [cs.AI]`.

[171] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. *Learning to reinforcement learn.* 2017. arXiv: `1611.05763 [cs.LG]`.

[172] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML).* 2017.

[173] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. *A Simple Neural Attentive Meta-Learner.* 2018. arXiv: `1707.03141 [cs.AI]`.

[174] Rein Houthooft, Richard Y. Chen, Phillip Isola, Bradly C. Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. *Evolved Policy Gradients.* 2018. arXiv: `1802.04821 [cs.LG]`.

[175] Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. *Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables.* 2019. arXiv: `1903.08254 [cs.LG]`.

[176] Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A. Ortega, Yee Whye Teh, and Nicolas Heess. *Meta reinforcement learning as task inference.* 2019. arXiv: `1905.06424 [cs.LG]`.

[177] Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J. Smola. *Meta-Q-Learning.* 2020. arXiv: `1910.00125 [cs.LG]`.

[178] Steindor Saemundsson, Katja Hofmann, and Marc P. Deisenroth. "Meta Reinforcement Learning with Latent Variable Gaussian Processes". In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI).* Acceptance rate: 25. 2018.

[179] Bradly Stadie, Ge Yang, Rein Houthooft, Peter Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. "The Importance of Sampling in Meta-Reinforcement Learning". In: *Advances in Neural Information Processing Systems.* Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Vol. 31. Curran Associates, Inc., 2018.

[180] Jin Zhang, Jianhao Wang, Hao Hu, Yingfeng Chen, Changjie Fan, and Chongjie Zhang. *Learn to Effectively Explore in Context-Based Meta-RL.* June 2020.

[181] Evan Zheran Liu, Aditi Raghunathan, Percy Liang, and Chelsea Finn. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices.* 2021. arXiv: `2008.02790 [cs.LG]`.

[182] Lanqing Li, Rui Yang, and Dijun Luo. "Efficient Fully-Offline Meta-Reinforcement Learning via Distance Metric Learning and Behavior Regularization". In: *International Conference on Learning Representations.* 2021.

[183] Karol Arndt, Murtaza Hazara, Ali Ghadirzadeh, and Ville Kyrki. "Meta Reinforcement Learning for Sim-to-real Domain Adaptation". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020).

[184] Djordje Grbic and Sebastian Risi. *Safe Reinforcement Learning through Meta-learned Instincts.* 2020. arXiv: `2005.03233 [cs.LG]`.

[185] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. "A survey of robot learning from demonstration". In: *Robotics and autonomous systems* 57.5 (2009), pp. 469–483.

[186] Saurabh Arora and Prashant Doshi. "A survey of inverse reinforcement learning: Challenges, methods and progress". In: *arXiv preprint arXiv:1806.06877* (2018).

[187] Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, and Jan Peters. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[188] Jonathan Spencer, Sanjiban Choudhury, Matthew Barnes, Matthew Schmittle, Mung Chiang, Peter Ramadge, and Siddhartha Srinivasa. “Learning from Interventions: Human-robot Interaction as both Explicit and Immplicit Feedback”. In: *Robotics: Science and Systems (RSS)*. 2020.

[189] Gokul Swamy, Siddharth Reddy, Sergey Levine, and Anca D Dragan. “Scaled autonomy: Enabling human operators to control robot fleets”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 5942–5948.

[190] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. “Where Do You Think You’re Going?: Inferring Beliefs about Dynamics from Behavior”. In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[191] John Launchbury. “A Natural Semantics for Lazy Evaluation”. In: POPL ’93. Charleston, South Carolina, USA: Association for Computing Machinery, 1993, pp. 144–154.

[192] Dean A Pomerleau. “Efficient training of artificial neural networks for autonomous navigation”. In: *Neural Computation* 3.1 (1991).

[193] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. “Dynamical movement primitives: learning attractor models for motor behaviors”. In: *Neural computation* 25.2 (2013).

[194] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. “Probabilistic movement primitives”. In: *Advances in neural information processing systems*. 2013, pp. 2616–2624.

[195] Faraz Torabi, Garrett Warnell, and Peter Stone. “Behavioral Cloning from Observation”. In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. Stockholm, Sweden, July 2018.

[196] Pieter Abbeel and Andrew Y Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, p. 1.

[197] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. “Maximum entropy inverse reinforcement learning.” In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2008.

[198] Jonathan Ho and Stefano Ermon. “Generative adversarial imitation learning”. In: *Advances in Neural Information Processing Systems*. 2016.

[199] Daniel S Brown, Wonjoon Goo, and Scott Niekum. “Better-than-Demonstrator Imitation Learning via Automaticaly-Ranked Demonstrations”. In: *Conference on Robot Learning (CoRL)*. 2019.

[200] Justin Fu, Katie Luo, and Sergey Levine. “Learning Robust Rewards with Adversarial Inverse Reinforcement Learning”. In: *arXiv preprint arXiv:1710.11248* (2017).

[201] Daniel S Brown, Scott Niekum, Russell Coleman, and Ravi Srinivasan. “Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences”. In: *International Conference on Machine Learning*. 2020.

[202] Akanksha Saran, Elaine Schaertl Short, Andrea Thomaz, and Scott Niekum. “Understanding Teacher Gaze Patterns for Robot Learning”. In: ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, Oct. 2020, pp. 1247–1258.

[203] Hsiao-Yu Tung, Adam W Harley, Liang-Kang Huang, and Katerina Fragkiadaki. “Reward learning from narrated demonstrations”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7004–7013.

[204] Michael Laskey, Jonathan Lee, Wesley Yu-Shu Hsieh, Richard Liaw, Jeffrey Mahler, Roy Fox, and Ken Goldberg. “Iterative Noise Injection for Scalable Imitation Learning”. In: *Conference on Robot Learning (CoRL)*. Vol. abs/1703.09327. 2017. arXiv: `1703.09327`.

[205] Ashwin Balakrishna*, Brijen Thananjeyan*, Jonathan Lee, Felix Li, Arsh Zahed, Joseph E. Gonzalez, and Ken Goldberg. “On-Policy Robot Imitation Learning from a Converging Supervisor”. In: *Conference on Robot Learning (CoRL)*. PMLR. 2019.

[206] Michael Laskey, Sam Staszak, Wesley Hsieh, Jeffrey Mahler, Florian Pokorny, Anca Dragan, and Ken Goldberg. “SHIV: Reducing Supervisor Burden using Support Vectors for Efficient Learning from Demonstrations in High Dimensional State Spaces”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[207] Kshitij Judah, Alan Fern, and Thomas Dietterich. “Active imitation learning via state queries”. In: *Proceedings of the icml workshop on combining learning strategies to reduce label cost*. Citeseer. 2011.

[208] Snehal Jauhri, Carlos Celemin, and Jens Kober. “Interactive Imitation Learning in State-Space”. In: *arXiv preprint arXiv:2008.00524* (2020).

[209] Dorsa Sadigh, Anca D. Dragan, S. Shankar Sastry, and Sanjit A. Seshia. “Active Preference-Based Learning of Reward Functions”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2017.

[210] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. “Deep reinforcement learning from human preferences”. In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[211] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. “Reward learning from human preferences and demonstrations in Atari”. In: *Advances in Neural Information Processing Systems*. 2018.

[212] Malayandi Palan, Nicholas C. Landolfi, Gleb Shevchuk, and Dorsa Sadigh. “Learning Reward Functions by Integrating Human Demonstrations and Preferences”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2019.

[213] Erdem Bıyık, Malayandi Palan, Nicholas C Landolfi, Dylan P Losey, and Dorsa Sadigh. "Asking easy questions: A user-friendly approach to active reward learning". In: *arXiv preprint arXiv:1910.04365* (2019).

[214] Siddharth Reddy, Anca D Dragan, and Sergey Levine. "Shared autonomy via deep reinforcement learning". In: *Robotics: Science and Systems (RSS)* (2018).

[215] Linhai Xie, Sen Wang, Stefano Rosa, Andrew Markham, and Niki Trigoni. "Learning with Training Wheels: Speeding up Training with a Simple Controller for Deep Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[216] Andrey Kurenkov, Ajay Mandlekar, Roberto Martin-Martin, Silvio Savarese, and Animesh Garg. "AC-Teach: A Bayesian Actor-Critic Method for Policy Learning with an Ensemble of Suboptimal Teachers". In: *Conference on Robot Learning (CoRL)*. 2019.

[217] Ching-An Cheng Nolan Wagener Byron Boots. "Safe Reinforcement Learning Using Advantage-Based Intervention". In: *International Conference on Machine Learning (ICML)*. 2021.

[218] William Saunders, Girish Sastry, Andreas Stuhlmüller, and Owain Evans. "Trial without Error: Towards Safe RL with Human Intervention". In: *17th International Conference on Autonomous Agents and MultiAgent Systems* (2018).

[219] Fan Wang, Bo Zhou, Ke Chen, Tingxiang Fan, Xi Zhang, Jiangyong Li, Hao Tian, and Jia Pan. "Intervention Aided Reinforcement Learning for Safe and Practical Policy Optimization in Navigation". In: *Conference on Robot Learning (CoRL)*. 2018.

[220] Gregory Kahn, Pieter Abbeel, and Sergey Levine. "LaND: Learning to Navigate from Disengagements". In: *arXiv preprint arXiv:2010.04689*. 2020.

[221] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J. Grosz. "Interactive Teaching Strategies for Agent Training". In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.

[222] Erik Båvenstrand and Jakob Berggren. "Performance Evaluation of Imitation Learning Algorithms with Human Experts". In: *Technical Report KTH Royal Institute of Technology Sweden*. 2019.

[223] Jacob W Crandall, Michael A Goodrich, Dan R Olsen, and Curtis W Nielsen. "Validating human-robot interaction schemes in multitasking environments". In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 35.4 (2005), pp. 438–449.

[224] Jessie YC Chen and Michael J Barnes. "Human–agent teaming for multirobot control: A review of human factors issues". In: *IEEE Transactions on Human-Machine Systems* 44.1 (2014), pp. 13–29.

[225] Taylor Kessler Faulkner, Reymundo A Gutierrez, Elaine Schaertl Short, Guy Hoffman, and Andrea L Thomaz. "Active attention-modified policy shaping: socially interactive agents track". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019.

[226] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[227] B. K. Bose. "An adaptive hysteresis-band current control technique of a voltage-fed PWM inverter for machine drive system". In: *IEEE Transactions on Industrial Electronics* 37.5 (1990).

[228] Daniel Seita, Aditya Ganapathi, Ryan Hoque, Minho Hwang, Edward Cen, Ajay Kumar Tanwani, Ashwin Balakrishna, Brijen Thananjeyan, Jeffrey Ichnowski, Nawid Jamali, Katsu Yamane, Soshi Iba, John Canny, and Ken Goldberg. "Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[229] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5026–5033.

[230] Ryan Hoque, Daniel Seita, Ashwin Balakrishna, Aditya Ganapathi, Ajay Tanwani, Nawid Jamali, Katsu Yamane, Soshi Iba, and Ken Goldberg. "VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation". In: *Robotics: Science and Systems (RSS)*. 2020.

[231] Zdravko I. Botev, Dirk P. Kroese, Reuven Y. Rubinstein, and Faculty Of Industrial Engineering. *The Cross-Entropy Method for Optimization*. 2013.

[232] Atsushi Sakai, Daniel Ingram, Joseph Dinius, Karan Chawla, Antonin Raffin, and Alexis Paques. "PythonRobotics: a Python code collection of robotics algorithms". In: (2018).

[233] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

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

[235] Justin Fu, John Co-Reyes, and Sergey Levine. "EX2: Exploration with exemplar models for deep reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2017, pp. 2577–2587.

[236] Vitchyr Pong. *rlkit*. `https://github.com/vitchyr/rlkit`. 2019.

[237] Rishabh Jangir. *Overcoming exploration from demos*. `https://github.com/jangirrishabh/Overcoming-exploration-from-demos`. 2018.

[238] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust Region Policy Optimization". In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, 2015, pp. 1889–1897.

[239] Brijen Thananjeyan Ashwin Balakrishna. *Code for Recovery RL*. `https://github.com/abalakrishna123/recovery-rl`. 2021.

[240] Harshit Sikchi. *Code for Advantage Weighted Actor Critic*. `https://github.com/hari-sikchi/AWAC`. 2021.

[241] Thanard Kurutach. *Model-Ensemble Trust-Region Policy Optimization (ME-TRPO)*. `https://github.com/thanard/me-trpo`. 2019.

[242] Quan Vuong. *PyTorch implementation of PETS*. `https://github.com/quanvuong/handful-of-trials-pytorch`. 2020.

Part V

Appendices

Chapter 10

Appendix for Chapter 4

10.0.1 Policy Domain Expansion Strategy

Here we discuss another way in which the policy domain can be expanded when the safe set and value function are updated based on samples from the exploration policy. To approximately expand $\bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^{k}$, we can again solve the following 1-step trajectory optimization problem with $x_0^j = x_S^j$:

$$\begin{aligned}
\pi^j_{E,0:H'-1}(x^j_S) = &\underset{\pi_{0:H'-1}\in\Pi^{H'}}{\text{argmin}} \quad \underset{w^j_{0:H'-2}}{\mathbb{E}}\left[\sum_{i=0}^{H'-1} C^j_E(x^j_i, \pi_i(x^j_i))\right] \\
\text{s.t.} \quad & x^j_{i+1} = f(x^j_i, \pi_i(x^j_i), w_i)\ \forall i \in \{0, \dots, H'-1\} \\
& x^j_{H'} \in \bigcup_{k=0}^{j-1} \mathcal{SS}^k_{\mathcal{G}},\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \\
& x^j_{0:H'} \in \mathcal{X}^{H'+1},\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1}
\end{aligned} \tag{10.1}$$

For all $x^j_S \in \mathcal{SS}^{j-1}_{\mathcal{G}}$, the states

$$\bigcup_{k=0}^{H'} \mathcal{R}_k^{\pi^j_{E,0:H'-1}}(x^j_S) \cup \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{R}_{H'}^{\pi^j_{E,0:H'-1}}(x^j_S)) \tag{10.2}$$

are added to $\mathcal{SS}^j_{\mathcal{G}}$. The second union is included to define the value function for the composition of π^j and $\pi^j_{E,0:H'-1}$. This is analogous to running the exploration policy followed by running the task-directed policy π^j. Denoting the safe set where π^j is executed as

$\mathcal{SS}_{\mathcal{G}}^{\pi^j} = \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{R}_{H'}^{\pi^j_{E,0:H'-1}}(\mathcal{SS}_{\mathcal{G}}^{j-1})) \cup \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{SS}_{\mathcal{G}}^{j-1})$, we redefine $L_{\mathcal{G}}^{\pi^j}$ as:

$$\begin{cases} 0 & x \in \mathcal{G} \\ +\infty & x \notin \mathcal{SS}_{\mathcal{G}}^j \\ \mathop{\mathbb{E}}_{w}\left[C(x, \pi^j(x)) + L_{\mathcal{G}}^{\pi^j}(f(x, \pi^j(x), w))\right] & x \in \mathcal{SS}_{\mathcal{G}}^{\pi^j} \setminus \mathcal{G} \\ \mathop{\mathbb{E}}_{w}\left[C(x, \pi^j_{E,0:H'-1}(x)) + L_{\mathcal{G}}^{\pi^j}(x'_E)\right] & x \in \mathcal{SS}_{\mathcal{G}}^j \setminus \mathcal{SS}_{\mathcal{G}}^{\pi^j} \end{cases} \tag{10.3}$$

where $x'_E = f(x, \pi^j_{E,0:H'-1}(x), w)$. This means that trajectories from the exploration policy can spend more time outside of the safe set. In either case, the safe set remains robust control invariant.

Thus, each iteration j is split into two phases. In the first phase, π^j is executed and in the second phase, $\pi^j_{E,0:H'-1}$ is executed. This procedure provides a simple algorithm to expand the policy's domain $\mathcal{F}_{\mathcal{G}}^j$ while still maintaining its theoretical properties.

10.1 Properties of ABC-LMPC

In this section, we study the properties of the policy constructed in Section 2.5. For analysis, we will assume a fixed goal set $\mathcal{G}$, but note that if the goal set is changed at some iteration, the same properties still apply to the new goal set $\mathcal{H}$ by the same proofs, because all of the same assumptions hold for $\mathcal{H}$.

Lemma 10.1.1. ***Recursive Feasibility:*** *Consider the closed-loop system* (2.11) *and* (2.12)*. Let the safe set* $\mathcal{SS}_{\mathcal{G}}^j$ *be defined as in* (2.5)*. If assumptions 2.3.1-2.4.1 hold and* $x_0^j \in \mathcal{F}_{\mathcal{G}}^j$*, then the policy induced by optimizing* (2.11) *and* (2.12) *is feasible almost surely for* $t \geq 0$ *and* $j \geq 0$*, i.e.,*

$$\mathop{\mathbb{E}}_{w^j_{0:H-1}}[J^j_{t\to t+H}(x^j_t)] < \infty, \forall t, j \geq 0.$$

Proof of Chapter 10.1.1: We proceed by induction. By assumption 2.4.1, $J^0_{0\to H}(x^j_0) < \infty$. By the definition of $V_{\mathcal{G}}^{\pi^j}$ and $\mathcal{F}_{\mathcal{G}}^j$, $J^j_{0\to H}(x^j_0) < \infty$. Let $J^j_{t\to t+H}(x^j_t) < \infty$ for some $t \in \mathbb{N}$. In the following expressions, we do not explicitly write the MPC problem constraints for clarity. Conditioning on the random variable x^j_t:

$$J^j_{t\to t+H}(x^j_t) = \tag{10.4}$$

$$\underset{w^j_{t:t+H-1}}{\mathbb{E}}\left[\sum_{k=0}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + V_{\mathcal{G}}^{\pi^{j-1}}(x^j_{t+H|t})\right] \tag{10.5}$$

$$= C(x^j_t, \pi^{*,j}_{t|t}(x^j_t)) \tag{10.6}$$

$$+ \mathbb{E}_{w^j_{t:t+H-1}}\left[\sum_{k=1}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + V_{\mathcal{G}}^{\pi^{j-1}}(x^j_{t+H|t})\right] \tag{10.7}$$

$$= C(x^j_t, \pi^{*,j}_{t|t}(x^j_t))$$

$$\begin{aligned} &+ \underset{w^j_{t:t+H}}{\mathbb{E}}\left[\sum_{k=1}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + C(x^j_{t+H|t}, \pi^l(x^j_{t+H|t}))\right. \\ &\left. + V_{\mathcal{G}}^{\pi^{j-1}}(x^j_{t+H+1|t})\right],\ l \in [j-1] \end{aligned} \tag{10.8}$$

$$\geq C(x^j_t, \pi^{*,j}_{t|t}(x^j_t))$$

$$\begin{aligned} &+ \underset{w^j_t}{\mathbb{E}}\left[\min_{\pi_{t+1:t+H|t+1}} \underset{w^j_{t+1:t+H}}{\mathbb{E}}\left[\sum_{k=1}^{H-1} C(x^j_{t+k|t+1}, \pi_{t+k|t+1}(x^j_{t+k|t+1}))\right.\right. \\ &+ C(x^j_{t+H|t+1}, \pi_{t+H|t+1}(x^j_{t+H|t+1})) \\ &\left.\left. + V_{\mathcal{G}}^{\pi^{j-1}}(x^j_{t+H+1|t+1})\right]\right] \end{aligned} \tag{10.9}$$

$$= C(x^j_t, \pi^j(x^j_t)) + \underset{w^j_t}{\mathbb{E}}\left[J^j_{t+1\to t+H+1}(x^j_{t+1})|x^j_t\right] \tag{10.10}$$

Equation 10.5 follows from the definition in 2.11, equation 10.8 follows from the definition of $V_{\mathcal{G}}^{\pi_{j-1}}$, which is defined as a point-wise minimum over $\left(L_{\mathcal{G}}^{\pi^l}\right)_{l=0}^{j-1}$. We take a function $L_{\mathcal{G}}^{\pi^l}$ that is active at $x^j_{t+H|t}$ and apply its definition to expand it and then replace $L_{\mathcal{G}}^{\pi^l}$ with $V_{\mathcal{G}}^{\pi^{j-1}}$ in the expansion. The inner expectation in equation 10.9 conditions on the random variable x^j_{t+1}, and the outer expectation integrates it out. The inequality in 10.9 follows from the fact that $[\pi^{*,j}_{t+1|t}, \ldots, \pi^{*,j}_{t+H-1|t}, \pi^{j-1}]$ is a possible solution to (10.9). Equation 10.10 follows from the definition in equation 2.11.

We have shown that $J^j_{t\to t+H}(x^j_t) < \infty \implies \underset{w^j_t}{\mathbb{E}}\left[J^j_{t+1\to t+H+1}(x^j_{t+1|t})\right] < \infty$. So:

$$\underset{w^j_{0:t-1}}{\mathbb{E}}\left[J^j_{t\to t+H}(x^j_t)\right] < \infty \implies \underset{w^j_{0:t-1}}{\mathbb{E}}\left[\underset{w^j_t}{\mathbb{E}}\left[J^j_{t+1\to t+H+1}(x^j_{t+1|t})\right]\right] \tag{10.11}$$

$$= \underset{w^j_{0:t}}{\mathbb{E}}\left[J^j_{t+1\to t+H+1}(x^j_{t+1})\right] < \infty \tag{10.12}$$

By induction, $\underset{w^j_{0:t-1}}{\mathbb{E}}[J^j_{t\to t+H}(x^j_t)] < \infty \ \forall t \in \mathbb{N}$. Therefore, the policy is feasible at iteration j. □

Chapter 10.1.1 shows that the policy is guaranteed to satisfy state-space constraints for all timesteps t in all iterations j given the definitions and assumptions presented above. Equivalently, the expected planning cost of the policy is guaranteed to be finite. The following lemma establishes convergence in probability to the goal set given initialization within the policy domain.

Lemma 10.1.2. ***Convergence in Probability:*** *Consider the closed-loop system defined by* (2.11) *and* (2.12)*. Let the sampled safe set* $\mathcal{SS}^j_{\mathcal{G}}$ *be defined as in* (2.5)*. Let assumptions 2.3.1-2.4.1 hold and* $x^j_0 \in \mathcal{F}^j_{\mathcal{G}}$*. If the closed-loop system converges in probability to* $\mathcal{G}$ *at iteration* 0*, then it converges in probability at all subsequent iterations. Stated precisely, at iteration* j*:* $\lim_{t\to\infty} P(x^j_t \notin \mathcal{G}) = 0$.

Proof of Chapter 10.1.2: By Chapter 10.1.1 and Assumption 2.3.1, $\forall L \in \mathbb{N}$,

$$\underset{w^j_{1:L-1}}{\mathbb{E}}\left[\sum_{k=0}^{L-1} C(x^j_k, \pi^j(x^j_k)) + J^j_{L\to L+H}(x^j_L)\right] \le J^j_{0\to H}(x^j_0) \tag{10.13}$$

$$\implies \underset{w^j_{1:L-1}}{\mathbb{E}}\left[J^j_{L\to L+H}(x^j_L)\right] \le J^j_{0\to H}(x^j_0) - \tag{10.14}$$

$$\underset{w^j_{1:L-1}}{\mathbb{E}}\left[\sum_{k=0}^{L-1} C(x^j_k, \pi^j(x^j_k))\right] \tag{10.15}$$

$$\le J^j_{0\to H}(x^j_0) - \epsilon \sum_{k=0}^{L-1} P(x^j_k \notin \mathcal{G}) \tag{10.16}$$

Line 10.16 follows from rearranging 10.13 and applying assumption 2.3.1. Because $\mathcal{G}$ is robust control invariant by assumption 2.3.2, $x_t \in \mathcal{G} \implies x_{t+k} \in \mathcal{G} \ \forall k \ge 0$. Now, assume $\lim_{k\to\infty} P(x^j_k \notin \mathcal{G})$ does not exist or is nonzero. This implies that $P(x^j_k \notin \mathcal{G}) \ge \delta > 0$ infinitely many times. By the Archimedean principle, the RHS of 10.16 can be driven arbitrarily negative, which is impossible. By contradiction, $\lim_{k\to\infty} P(x^j_k \notin \mathcal{G}) = 0$. □

Theorem 10.1.1. ***Iterative Improvement:*** *Consider system* (2.1) *in closed-loop with* (2.11) *and* (2.12). *Let the sampled safe set* $\mathcal{SS}^j$ *be defined as in* (2.5). *Let assumptions 2.3.1-2.4.1 hold, then the expected cost-to-go* (2.6) *associated with the control policy* (2.12) *is non-increasing in iterations for a fixed start state. More formally:*

$$\forall j \in \mathbb{N},\ x_0^j \in \mathcal{F}_{\mathcal{G}}^j,\ x_0^{j+1} \in \mathcal{F}_{\mathcal{G}}^{j+1} \implies J^{\pi^j}(x_0^j) \geq J^{\pi^{j+1}}(x_0^{j+1})$$

Furthermore, $\{J^{\pi^j}(x_0^j)\}_{j=0}^{\infty}$ *is a convergent sequence.*

Proof of Chapter 10.1.1: Let $j \in \mathbb{N}$

$$J_{0\to H}^j(x_0) \geq C(x_0, u_0) + \mathop{\mathbb{E}}_{w_0^j}\left[J_{1\to H+1}^j(x_1^j)\right] \tag{10.17}$$

$$\geq \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty} \mathop{\mathbb{E}}_{w_{0:t-1}^j}\left[J_{t\to t+H}^j(x_t^j)\right] \tag{10.18}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] \tag{10.19}$$

$$+ \lim_{t\to\infty} \mathop{\mathbb{E}}_{\mathbb{1}\{x_t^j \notin \mathcal{G}\}}\left[\mathop{\mathbb{E}}_{w_{0:t-1}^j}\left[J_{t\to t+H}^j(x_t)|\mathbb{1}\{x_t^j \notin \mathcal{G}\}\right]\right] \tag{10.20}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] \tag{10.21}$$

$$+ \lim_{t\to\infty} \mathop{\mathbb{E}}_{w_{0:t-1}^j}\left[J_{t\to t+H}^j(x_t^j)|x_t^j \notin \mathcal{G}\right] P(x_t^j \notin \mathcal{G}) \tag{10.22}$$

$$\geq \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty} \epsilon P(x_t^j \notin \mathcal{G}) \tag{10.23}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] = J^{\pi^j}(x_0) \tag{10.24}$$

Equations 10.17 and 10.18 follow from repeated application of Chapter 10.1.1 (10.10). Equation 10.20 follows from iterated expectation, equation 10.22 follows from the cost function assumption 2.3.1. Equation 10.23 follows again from assumption 2.3.1 (incur a cost of at least ϵ for not being at the goal at time t). Then, Equation 10.24 follows from Chapter 10.1.2.

Using the above inequality with the definition of $J^{\pi^j}(x_0)$,

$$J^j_{0\to H}(x_0) \geq J^{\pi^j}(x_0) = \mathop{\mathbb{E}}_{w^j_{0:H-1}}\left[\sum_{t=0}^{H-1} C(x^j_t, \pi^j(x_t)) + V^{\pi^j}_{\mathcal{G}}(x^j_H)\right] \tag{10.25}$$

$$\geq \mathop{\mathbb{E}}_{w^j_{0:H-1}}\left[\sum_{t=0}^{H-1} C(x^j_t, \pi^{*,j}_{t|0}(x_{t|0})) + V^{\pi^j}_{\mathcal{G}}(x_{H|0})\right] = J^{j+1}_{0\to H}(x_0) \tag{10.26}$$

$$\geq J^{\pi^{j+1}}(x_0) \tag{10.27}$$

Equation 10.25 follows from equation 10.24, equation 10.26 follows from taking the minimum over all possible H-length sequences of policies in the policy class Π. Equation 10.27 follows from equation 10.24. By induction, this proves the theorem.

Note that this also implies convergence of $(J^{\pi^j}(x_0))_{j=0}^{\infty}$ by the Monotone Convergence Theorem. □

Chapter 10.1.1 extends prior results [16], which guarantee robust iterative improvement for stochastic linear systems with convex costs and convex constraint sets. Here we show iterative improvement in *expectation* for ABC-LMPC for stochastic nonlinear systems with costs as in Assumption 2.3.1. The following result implies that the policy domain is non-decreasing.

Lemma 10.1.3. ***Policy domain expansion:*** *The domain of π^j is an non-decreasing sequence of sets: $\mathcal{F}^j_{\mathcal{G}} \subseteq \mathcal{F}^{j+1}_{\mathcal{G}}$.*

Proof of Chapter 10.1.3: The proof is identical to [16]. Because $\mathcal{SS}^j_{\mathcal{G}}$ is an increasing sequence of sets, $\mathcal{F}^j_{\mathcal{G}}$ is also an increasing sequence of sets by definition. □

10.2 Practical Implementation

ABC-LMPC alternates between two phases at each iteration: the first phase performs the task by executing π^j and the second phase runs the exploration policy $\pi^j_{E,0:H'-1}$. Only data from π^j is added to an approximation of $\mathcal{SS}^j_{\mathcal{G}}$, on which the value function L^{π^j} is fit, but in principle, data from $\pi^j_{E,0:H'-1}$ can also be used. Although the task (2.11) and exploration (2.14) objectives are generally intractable, we present a simple algorithm which introduces sample-based approximations to expand the policy's domain $\mathcal{F}^j_{\mathcal{G}}$ while approximately maintaining theoretical properties in practice. Here, we describe how each component in the policy design is implemented and how optimization is performed. See Appendix 10.2 for further implementation details.

10.2.1 Sample-Based Safe Set

In practice, as in [16], we approximate the safe set $\mathcal{SS}^j_{\mathcal{G}}$ using samples from the closed loop system defined by (2.11) and (2.12). To do this, we collect R closed-loop trajectories at iteration j, each of length T as in [16] where T is the task horizon.

Thus, given the ith disturbance realization sequence collected at iteration j, given by $\mathbf{w}^j_i = [w^j_{0,i}, \dots w^j_{T,i}]$, we define the closed loop trajectory associated with this sequence as in [16]: $\mathbf{x}^j(\mathbf{w}^j_i) = \left[x^j_0(\mathbf{w}^j_i), \dots, x^j_T(\mathbf{w}^j_i)\right]$. As in [16], we note that $x^j_k(\mathbf{w}^j_i) \in \mathcal{R}^{\pi^j}_k(x^j_0)$, so R rollouts from the closed-loop system provides a sample-based approximation to $\mathcal{R}^{\pi^j}_k(x^j_0)$ as follows: $\tilde{\mathcal{R}}^{\pi^j}_k(x^j_0) = \bigcup_{i=1}^{R} x^j_k(\mathbf{w}^j_i) \subseteq \mathcal{R}^{\pi^j}_k(x^j_0)$. Similarly, we can define a sample-based approximation to the safe set as follows: $\tilde{\mathcal{SS}}^j_{\mathcal{G}} = \left\{\bigcup_{k=0}^{\infty} \tilde{\mathcal{R}}^{\pi^j}_k(x^j_0) \bigcup \mathcal{G}\right\}$.

While $\tilde{\mathcal{SS}}^j_{\mathcal{G}}$ is not robust control invariant, with sufficiently many trajectory samples (i.e. R sufficiently big), this approximation becomes more accurate in practice [16]. To obtain a continuous approximation of the safe set for planning, we use the same technique as [24], and fit density model $\rho^{\mathcal{G}}_{\alpha}$ to $\bigcup_{k=0}^{j-1} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$ and instead of enforcing the terminal constraint by checking if $x_{t+H} \in \bigcup_{k=0}^{j-1} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$, ABC-LMPC instead enforces that $\rho^{\mathcal{G}}_{\alpha}(x_{t+H}) > \delta$, where α is a kernel width parameter. We implement a tophat kernel density model using a nearest neighbors classifier with tuned kernel width α and use $\delta = 0$ for all experiments. Thus, all states within Euclidean distance α from the closest state in $\bigcup_{k=0}^{j-1} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$ are considered safe under $\rho^{\mathcal{G}}_{\alpha}$.

10.2.2 Start State Expansion Strategy

To provide a sample-based approximation to the procedure from Section 2.5.2, we sample states x^j_S from $\bigcup_{k=0}^{j} \tilde{\mathcal{SS}}^k$ and execute $\pi^j_{E,0:H'-1}$ for R trajectories of length H', which approximate $\mathcal{M}$. We repeat this process until an x^j_S is found such that all R sampled trajectories satisfy the terminal state constraint that $x^j_{H'} \in \bigcup_{k=0}^{j} \tilde{\mathcal{SS}}^k_{\mathcal{G}}$ (Section 2.5.2). Once such a state is found, a state is sampled from the last H steps of the corresponding trajectories to serve as the start state for the next iteration, which approximates sampling from $\mathcal{M}_H$. We utilize a cost function which encourages policy domain expansion towards a specific desired start state x^*, although in general any cost function can be used. This cost function is interesting because it enables adaptation of a learning MPC policy to desired specifications while maintain policy feasibility. Precisely, we optimize a cost function which simply measures the discrepancy between a given state in a sampled trajectory and x^*, ie. $C^j_E(x, u) = D(x, x^*)$. This distance measure can be tuned on a task-specific basis based on the appropriate distance measures for the domain (Section 10.3.3). However, we remark that this technique requires: (1) an appropriate distance function $D(\cdot, \cdot)$ and (2) a reverse path from the goal to the start state, that may differ from the optimal forward path, along which the goal is robustly reachable.

10.2.3 Goal Set Transfer

We practically implement the goal set transfer strategy in Section 2.4.3 by fitting a new density model $\rho_{\alpha}^{\mathcal{H}}$ on the prefixes of prior trajectories that intersect some new user-specified goal set $\mathcal{H}$. If $\mathcal{H}$ is chosen such that $\tilde{\mathcal{SS}}^{j}_{\mathcal{H}}$ contains many states, the policy can seamlessly transfer to $\mathcal{H}$. If this is not the case, the policy domain for $\mathcal{H}$ must be expanded from $\mathcal{H}$ until it intersects many trajectories in the original domain.

10.2.4 ABC-LMPC Optimization Procedure

As in prior work on MPC for nonlinear control [24, 25], we solve the MPC optimization problem in (2.11) over sampled open loop sequences of controls using the cross entropy method (CEM) [231]. In practice, we implement the terminal safe set constraints and state-space constraints in (2.11) and (2.14) by imposing a large cost on sampled action sequences which violate constraints when performing CEM.

As in [24], we sample a fixed population size of action sequences at each iteration of CEM from a truncated Gaussian. These action sequences are simulated over a known model of the system dynamics and then the sampling distribution for the next iteration is updated based on the lowest cost sampled trajectories. For the cross entropy method we build off of the implementation in [25]. Precisely, at each timestep in a trajectory, a conditional Gaussian is initialized with the mean based on the final solution for the previous timestep and some fixed variance. Then, at each iteration of CEM, pop_size action sequences of plan_hor length are sampled from the conditional Gaussian, simulated over a model of the system dynamics, and then the num_elites samples with the lowest sum cost are used to refit the mean and variance of the conditional Gaussian distribution for the next iteration of CEM. This process is repeated num_iters times. The sum cost of an action sequence is computed by summing up the task cost function at each transition in the resulting simulated trajectory and then adding a large penalty for each constraint violating state in the simulated trajectory and an additional penalty if the terminal state in the simulated trajectory does not have sufficient density under $\rho_{\mathcal{G}}$. For all experiments, we add a $1e6$ penalty for violating terminal state constraints and a $1e8$ penalty for violating task constraints. In practice to accelerate domain expansion to x^*, when selecting initial states x^j_S from $\bigcup_{k=0}^{j} \tilde{\mathcal{SS}}^{k}$, we sort states in the safeset under $C^j_E(x)$ and use this to choose x^j_S close to x^* under $C^j_E(x)$. Note that this choice does not impact any of the theoretical guarantees.

We use a probabilistic ensemble of 5 neural networks to approximate $L^{\pi^j}_{\mathcal{G}}(x)$ as in [24]. In contrast to [24], a separate $L^{\pi^j}_{\mathcal{G}}(x)$ is fit using data from each iteration instead of fitting a single function approximator on all data. We utilize Monte Carlo estimates of the cost-to-go values when fitting $L^{\pi^j}_{\mathcal{G}}(x)$. Each element of the ensemble outputs the parameters of a conditional axis-aligned Gaussian distribution and are trained on bootstrapped samples from the training dataset using a maximum likelihood [25]. We represent each member of the probabilistic ensemble of neural networks used to approximate $L^{\pi^j}(x)$ with a neural network with 3 hidden layers, each with 500 hidden units. We use swish activations, and update weights using the Adam Optimizer with learning rate 0.001. We use 10 epochs to learn the weights for $L^{\pi^j}(x)$.

To implement the start state expansion strategy in Section 10.2.2, we again perform trajectory optimization using the cross entropy method and for each experiment use the same pop_size, num_elites, num_iters parameters as for solving the MPC problem. Costs for action sequences are computed by summing up $C_E^j(x)$ evaluated at each state x in the corresponding simulated trajectory, and the same mechanism is used for enforcing the terminal state constraint and task constraints as for solving the MPC problem.

10.3 Experiments

We evaluate whether ABC-LMPC can enable (1) iterative improvement in expected performance for stochastic nonlinear systems, (2) adaptation to new start states and (3) transfer to new goal sets on 3 simulated continuous control domains. In Section 10.3.1 we describe the experimental domains, in Section 10.3.2, we evaluate the policy with fixed start states and goal sets, in Section 10.3.3, we expand the policy domain iteratively toward a desired start state far from the goal set, in Section 10.3.4, we switch the goal set during learning, and finally in Section 10.3.5 we utilize both start state expansion and the goal set transfer technique to control a pendulum to an upright position. In all experiments, we use $C(x, u) = \mathbb{1}\{x \notin \mathcal{G}\}$ as in [24]. Note that for this cost function, the maximum trajectory cost is the task horizon T, and the resulting objective corresponds to minimum time optimal control. We include comparisons to the minimum trajectory cost achieved by the state-of-the-art demonstration augmented model-based reinforcement learning algorithm, SAVED [24] after 10 iterations of training to evaluate the quality of the learned policy. For all experiments, we use $R = 5$ closed-loop trajectories from the current policy to estimate $\tilde{\mathcal{SS}}_{\mathcal{G}}^j$ and perform start state expansion. Experimental domains have comparable stochasticity to those in [16].

10.3.1 Experimental Domains

Point Mass Navigation: We consider a 4-dimensional (x, y, v_x, v_y) navigation task as in [24], in which a point mass is navigating to a goal set (a unit ball centered at the origin unless otherwise specified). The agent exerts force (f_x, f_y), $\|(f_x, f_y)\| \leq 1$, in each cardinal direction and experiences drag coefficient ψ. We introduce truncated Gaussian process noise $z_t \sim \mathcal{N}(0, \sigma^2 I)$ in the dynamics with domain $[-\sigma, \sigma]$. We include a large obstacle in the center of the environment that the robot must navigate around to reach the goal. While this task has linear dynamics, the algorithm must consider non-convex state space constraints and stochasticity. We use $\psi = 0.2$ and $\sigma = 0.05$ in all experiments in this domain. Demonstration trajectories are generated by guiding the robot past the obstacle along a very suboptimal hand-tuned trajectory for the first half of the trajectory before running LQR with clipped actions on a quadratic approximation of the true cost. Gaussian noise is added to the demonstrator policy. The task horizon is set to $T = 50$.

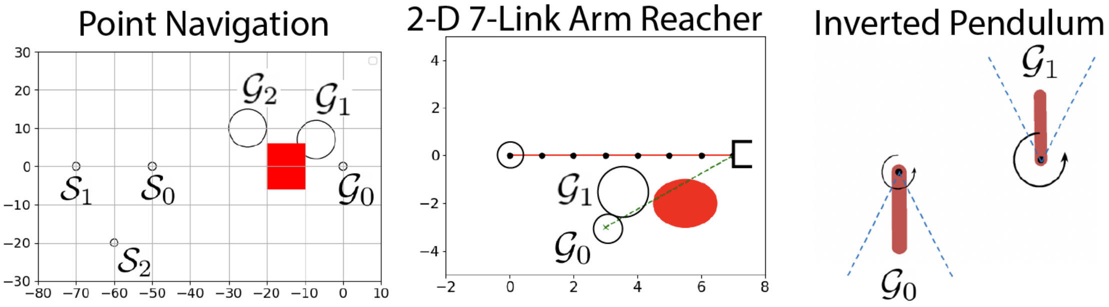


Figure 10.1: **Experimental Domains:** We evaluate ABC-LMPC on three stochastic domains: a navigation domain with an obstacle, a 2D 7-link arm reacher domain with an obstacle, and an inverted pendulum domain. In the first two domains, suboptimal demonstrations are provided, while no demonstrations are provided for the inverted pendulum task.

7-Link Arm Reacher: Here, we consider a 2D kinematic chain with 7 joints where the agent provides commands in delta joint angles. We introduce truncated Gaussian process noise $z_t \sim \mathcal{N}(0, \sigma^2 I)$ in the dynamics with domain $[-\sigma, \sigma]$ and build on the implementation from [232]. The goal is to control the end effector position to a 0.5 radius circle in $\mathbb{R}^2$ centered at $(3, -3)$. We use $\sigma = 0.03$ for all experiments. The state space consists of the 7 joint angles. Each link is of 1 unit in length and the goal is to control the end effector position to a 0.5 radius circle in $\mathbb{R}^2$ centered at $(3, -3)$. We do not model self-collisions but also include a circular obstacle of radius 1 in the environment which the kinematic chain must navigate around. Collisions with the obstacle are checked by computing the minimum distance between each link in the kinematic chain and the center of the circular obstacle and determining whether any link has a minimum distance from the center of the obstacle that is less than the radius of the obstacle. The task horizon is set to $T = 50$. We build on the implementation provided through [232].

Inverted Pendulum: This environment is a noisy inverted pendulum task adapted from OpenAI Gym [233]. We introduce truncated Gaussian process noise in the dynamics with standard deviation $\sigma = 0.5$ for all experiments. The robot consists of a single link and can exert a torque to rotate it. The state space consists of the angle and angular velocity of the pendulum. Note that there are only 2 stable orientations, the upright orientation and downward orientation for this task, and thus for a goal set to be robust control invariant, it will likely need to be defined around the neighborhood of these orientations. The task horizon is set to $T = 40$. We define $\mathcal{G}_1$ as the goal set centered around the downward orientation and $\mathcal{G}_2$ as the goal set centered around the upright orientation. Precisely, inclusion in $\mathcal{G}_1$ is determined by determining whether the orientation of the pendulum is within 45 degrees of the downward orientation. Similarly, inclusion in $\mathcal{G}_2$ is determined by determining whether the orientation of the pendulum is within 45 degrees of the upward orientation.

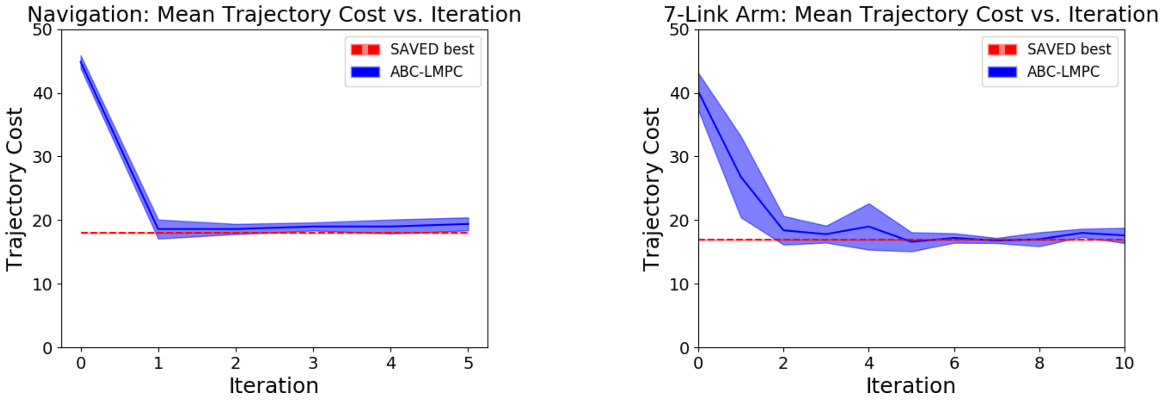


Figure 10.2: **Fixed Start, Single Goal Set Experiments:** Learning curves for ABC-LMPC averaged over $R = 5$ rollouts per iteration on simulated continuous control domains when the start state and goal set is held fixed during learning. Performance of the demonstrations is shown at iteration 0, and the policy performance is shown thereafter. **Point Mass Navigation:** The policy immediately improves significantly upon the demonstration performance within 1 iteration, achieving a mean trajectory cost of around 20 while demonstrations have mean trajectory cost of 42.58. **7-Link Arm Reacher:** The policy significantly improves upon the demonstrations, achieving a final trajectory cost of around 18 while demonstrations achieve a mean trajectory cost of 37.77. In all experiments, the policy quickly converges to the best cost produced by SAVED.

10.3.2 Fixed Start and Goal Conditions

We first evaluate ABC-LMPC on the navigation and reacher environments with a fixed start state and goal set. In the navigation domain, the robot must navigate from $\mathcal{S}_0 = (-50, 0, 0, 0)$ to the origin ($\mathcal{G}_0$) while in the reacher domain, the agent must navigate from a joint configuration with the end effector at $(7, 0)$ to one with the end effector at $(3, -3)$ ($\mathcal{G}_1$). For the navigation domain, we use popsize = 400, num_elites = 40, cem_iters = 5, and plan_hor = 15 and $\alpha = 2$ for the kernel width parameter for density model $\rho_\alpha^{\mathcal{G}}$. For the reacher domain, we use popsize = 400, num_elites = 40, cem_iters = 5, and plan_hor = 15 and $\alpha = 0.5$ for the kernel width parameter for density model $\rho_\alpha^{\mathcal{G}}$. The policy rapidly and significantly improves upon the 100 provided demonstrations in both domains (Figure 10.2). We compare ABC-LMPC to SAVED, which is provided with the same demonstrations as ABC-LMPC, using the implementation [24]. We use kernel width parameters $\alpha = 3$ and $\alpha = 0.5$ for the navigation and reacher domains respectively and find that ABC-LMPC achieves comparable cost to SAVED for both tasks and never violates constraints during learning.

10.3.3 Start State Expansion

ABC-LMPC is now additionally provided a target start state which is initially outside its domain and learns to iteratively expand its domain toward the desired start state. We report

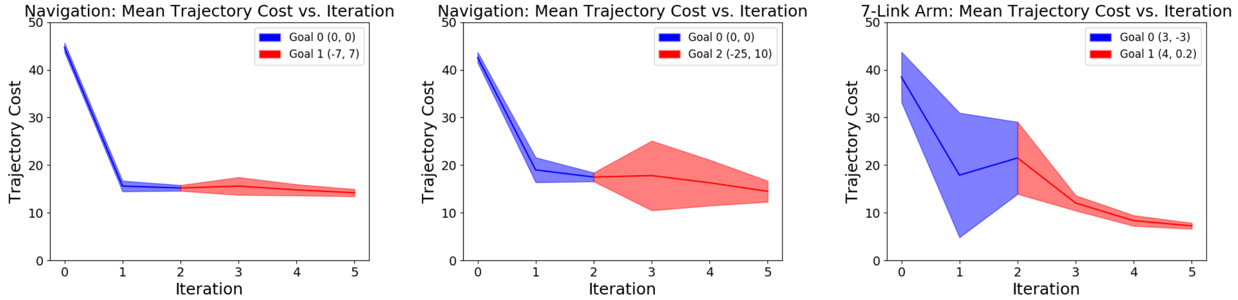


Figure 10.3: **Goal Set Transfer Learning:** In this experiment, the goal set is switched to to a new goal set at iteration 3 and we show a learning curve which indicates performance on both the first goal (blue) and new goal (red). The policy is re-trained as in Section 10.2.3 to stabilize to the new goal. The policy immediately is able to perform the new task and never hits the obstacle. Results are plotted over $R = 5$ policy rollouts per iteration.

the sequence of achieved start states over iterations in addition to the mean and standard deviation trajectory cost. ABC-LMPC is able to maintain feasibility throughout learning and achieve comparable performance to SAVED at the final start state when SAVED is supplied with 100 demonstrations from the desired state. To ensure that results are meaningful, we specifically pick desired start states such that given 100 demonstrations from the original start state, ABC-LMPC is never able to accomplish the task after 30 iterations of learning. A different $C_E(x, u)$ is used for start state expansion based on an appropriate distance metric for each domain. For all start state expansion experiments, we utilize $H' = H - 5$ for the trajectory optimization horizon for start state expansion. Otherwise, we use the same optimization parameters as in Section 10.3.2 except for plan_hor, which we set to 20 for all domains.

We first consider a navigation task where 100 suboptimal demonstrations are supplied from $(-25, 0, 0, 0)$ with average trajectory cost of 44.76. The goal is to expand the policy domain in order to navigate from start states $\mathcal{S}_1 = (-70, 0, 0, 0)$ and $\mathcal{S}_2 = (-60, -20, 0, 0)$. $C_E(x, u)$ measures the Euclidean distance between the positions of x and those of the desired start state. After 20 iterations, the policy reaches the desired start state while consistently maintaining feasibility during learning (Table 10.1).

We then consider a similar Reacher task using the same suboptimal demonstrations from Section 10.3.2. The desired start end effector position is $(-1, 0)$, and $C_E(x, u)$ measures the Euclidean distance between the end effector position of states x in optimized trajectories and that of the desired start state. Within 16 iterations of learning, the policy is able to start at the desired start state while maintaining feasibility during learning (Table 10.1). On both domains, the policy achieves comparable performance to SAVED when trained with demonstrations from that start state and the policy successfully expands its domain while rapidly achieving good performance at the new states. Constraints are never violated during learning for all experiments.

Table 10.1: **Start State Expansion Experiments:** Pointmass Navigation: Start State Expansion towards position $(-70, 0)$ (left) and $(-60, -20)$ (center). Here we see that ABC-LMPC is able to reach the desired start state in both cases while consistently maintaining policy feasibility throughout learning. Furthermore, the policy achieves competitive performance with SAVED, which achieves a minimum trajectory cost of 21 from $(-70, 0)$ and 23 from $(-60, -20)$; 7-link Arm Reacher: Here we expand the start state from that corresponding to an end effector position of $(7, 0)$ to that corresponding to an end effector position of $(-1, 0)$ (right). Again, we see that the policy consistently maintains feasibility during learning and achieves trajectory costs comparable to SAVED, which achieves a minimum trajectory cost of 24. The trajectory costs are presented in format: mean ± standard deviation over $R = 5$ rollouts.

Point Navigation (-70, 0)

Iteration	Start Pos (x, y)	Trajectory Cost
4	$(-42.3, 1.33)$	23.0 ± 0.89
8	$(-54.1, 0.08)$	22.8 ± 1.67
12	$(-61.2, 2.70)$	25.0 ± 2.37
16	$(-70.3, -0.26)$	32.6 ± 5.08
20	$(-70.4, 0.12)$	29.4 ± 2.33

Point Navigation (-60, -20)

Iteration	Start Pos (x, y)	Trajectory Cost
4	$(-42.6, -8.76)$	19.6 ± 4.22
8	$(-54.6, -14.2)$	25.6 ± 5.23
12	$(-58.8, -20.3)$	27.2 ± 12.0
16	$(-60.6, -20.2)$	21.0 ± 0.63
20	$(-60.5, -19.6)$	22.4 ± 1.85

7-Link Reacher

Iteration	Start EE Position	Trajectory Cost
4	$(-1.28, -0.309)$	31.6 ± 8.04
8	$(-0.85, -0.067)$	30.8 ± 15.7
12	$(-0.95, -0.014)$	20.2 ± 1.83
16	$(-1.02, -0.023)$	19.4 ± 4.03

10.3.4 Goal Set Transfer

ABC-LMPC is trained as in Section 10.3.2, but after a few iterations, the goal set is changed to a new goal set that is in the policy domain. In the navigation domain, the robot is supplied a new goal set centered at $\mathcal{G}_1 = (-25, 10, 0, 0)$ or $\mathcal{G}_2 = (-7, 7, 0, 0)$ with radius 7 after 2 iterations of learning on the original goal set. We increase the radius so more prior trajectories can be reused for the new goal-conditioned value function. Results are shown in Figure 10.3 for both goal set transfer experiments. We also perform a goal set transfer experiment on the 7-link Reacher Task in which the robot is supplied a new goal set centered at $\mathcal{G}_1 = (4, 0.2)$ with radius 1 after 2 iterations of training. Results are shown in Figure 10.3. In both domains, ABC-LMPC seamlessly transfers to the new goal by leveraging prior experience to train a new set of value functions.

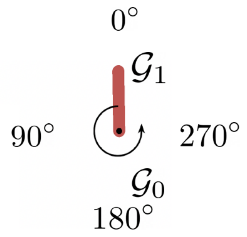
10.3.5 Inverted Pendulum Swing-Up Task

In this experiment, we incorporate both the start state optimization procedure and goal set transfer strategies to balance a pendulum in the upright position, but without any demonstrations. We utilize popsize = 600, num_elites = 40, cem_iters = 5, and plan_hor = 15. For experiments we utilize $\alpha = 2$ for the kernel width parameter for density model $\rho_\alpha^{\mathcal{G}}$ and $H' = H$ for the trajectory optimization horizon for start state expansion. We initialize the pendulum in the downward orientation ($\mathcal{G}_0$), and the goal of the task is to eventually stabilize the system to the upright orientation ($\mathcal{G}_1$). We iteratively expand the policy domain using the start state expansion strategy with initial goal $\mathcal{G}_0$ until the pendulum has swung up sufficiently close to the upright orientation. Once this is the case, we switch the goal set to $\mathcal{G}_1$ to stabilize to the upright position. The policy seamlessly transitions between the two goal sets, immediately transitioning to $\mathcal{G}_1$ while completing the task (convergence to either $\mathcal{G}_0$ or $\mathcal{G}_1$ within the task horizon) on all iterations (Table 10.2). $C_E(x, u)$ measures the distance between the pendulum's orientation and the desired start state's orientation.

10.4 Discussion and Future Work

We present a new algorithm for iteratively expanding the set of feasible start states and goal sets for an LMPC-based policy and provide theoretical guarantees on iterative improvement in expectation for non-linear systems under certain conditions on the cost function and demonstrate its performance on stochastic linear and nonlinear continuous control tasks. In future work, we will explore synergies with sample based motion planning to efficiently generate asymptotically optimal plans. We will also integrate the reachability-based domain expansion strategies of ABC-LMPC with model-based RL to learn safe and efficient policies when dynamics are learned from experience.

Table 10.2: **Pendulum Swing Up Experiment:** We iteratively expand the policy domain outward from a goal set centered around the downward orientation ($\mathcal{G}_0$) towards the upward orientation until the policy domain includes a goal set centered around the upward orientation ($\mathcal{G}_1$). Then, the goal set is switched to $\mathcal{G}_1$. The resulting policy maintains feasibility throughout and seamlessly transitions to $\mathcal{G}_1$. The trajectory costs are presented as: mean ± standard deviation over $R = 5$ rollouts. The upward orientation corresponds to a pendulum angle of 0° and the angle (degrees) increases counterclockwise from this position until 360°.



Iteration	Start Angle	Goal Set	Trajectory Cost
3	200.3	$\mathcal{G}_0$	30.2 ± 1.47
6	74.3	$\mathcal{G}_0$	35.0 ± 0.00
9	53.9	$\mathcal{G}_0$	34.4 ± 0.49
12	328.1	$\mathcal{G}_1$	36.0 ± 0.63
15	345.1	$\mathcal{G}_1$	13.8 ± 7.03
18	0.6	$\mathcal{G}_1$	0.00 ± 0.00

Chapter 11

Appendix for Chapter 5

11.1 Additional Experimental Details for SAVED and Baselines

For all experiments, we run each algorithm 3 times to control for stochasticity in training and plot the mean iteration cost vs. time with error bars indicating the standard deviation over the 3 runs. Additionally, when reporting task success rate and constraint satisfaction rate, we show bar plots indicating the median value over the 3 runs with error bars between the lowest and highest value over the 3 runs. Experiments are run on an Nvidia DGX-1 and on a desktop running Ubuntu 16.04 with a 3.60 GHz Intel Core i7-6850K, 12 core CPU and an NVIDIA GeForce GTX 1080. When reporting the iteration cost of SAVED and all baselines, any constraint violating trajectory is reported by assigning it the maximum possible iteration cost T, where T is the task horizon. Thus, any constraint violation is treated as a catastrophic failure. We plan to explore soft constraints as well in future work.

11.1.1 SAVED

Dynamics and Value Function

For all environments, dynamics models and value functions are each represented with a probabilistic ensemble of 5, 3 layer neural networks with 500 hidden units per layer with swish activations as used in Chua et al. [25]. To plan over the dynamics, the TS-∞ trajectory sampling method from [25] is used. We use 5 and 30 training epochs for dynamics and value function training when initializing from demonstrations. When updating the models after each training iteration, 5 and 15 epochs are used for the dynamics and value functions respectively. All models are trained using the Adam optimizer with learning rate 0.00075 and 0.001 for the dynamics and value functions respectively. Value function initialization is done by training the value function using the true cost-to-go estimates from demonstrations. However, when updated on-policy, the value function is trained using temporal difference error (TD-1) on a buffer containing all prior states. Since we use a probabilistic ensemble of neural networks to represent dynamics models and value functions, we built off of the provided implementation [234] of PETS in [25].

Constrained Exploration

Define states from which the system was successfully stabilized to the goal in the past as safe states. We train density model ρ on a fixed history of safe states, where this history is tuned based on the experiment. We have found that simply training on all prior safe states works well in practice on all experiments in this chapter. We represent the density model using kernel density estimation with a top-hat kernel. Instead of modifying δ for each environment, we set $\delta = 0$ (keeping points with positive density), and modify α (the kernel parameter/width). We find that this works well in practice, and allows us to speed up execution by using a nearest neighbors algorithm implementation from scikit-learn. We are experimenting with locality sensitive hashing, implicit density estimation as in Fu et al. [235], and have had some success with Gaussian kernels as well (at significant additional computational cost). The exploration strategy used by SAVED in navigation task 2 is illustrated in Figure 11.1.

11.1.2 Behavior Cloning

We represent the behavior cloning policy with a neural network with 3 layers of 200 hidden units each for navigation tasks and pick and place, and 2 layers of 20 hidden units each for the PR2 Reacher task. We train on the same demonstrations provided to SAVED and other baselines for 50 epochs.

11.1.3 PETSfD and PETSfD Dense

PETSfD and PETSfD Dense use the same network architectures and training procedure as SAVED and the same parameters for each task unless otherwise noted, but just omit the value function and density model ρ for enforcing constrained exploration. PETSfD uses a planning horizon that is long enough to complete the task, while PETSfD Dense uses the same planning horizon as SAVED.

11.1.4 SACfD

We use the rlkit implementation [236] of soft actor critic with the following parameters: batch size=128, discount=0.99, soft target $\tau = 0.001$, policy learning rate $= 3e-4$, Q function learning rate $= 3e-4$, and value function learning rate $= 3e-4$, batch size = 128, replay buffer size = 1000000, discount factor = 0.99. All networks are two-layer multi-layer perceptrons (MLPs) with 300 hidden units. On the first training iteration, only transitions from demonstrations are used to train the critic. After this, SACfD is trained via rollouts from the actor network as usual. We use a similar reward function to that of SAVED, with a reward of -1 if the agent is not in the goal set and 0 if the agent is in the goal set. Additionally, for environments with constraints, we impose a reward of -100 when constraints are violated to encourage constraint satisfaction. The choice of collision reward is ablated in section 11.4.2. This reward is set to prioritize constraint satisfaction over task success, which is consistent with the selection of β in the model-based algorithms considered.

11.1.5 OEFD

We use the implementation of OEFD provided by Jangir [237] with the following parameters: learning rate = 0.001, polyak averaging coefficient = 0.8, and L2 regularization coefficient = 1. During training, the random action selection rate is 0.2 and the noise added to policy actions is distributed as $\mathcal{N}(0, 1)$. All networks are three-layer MLPs with 256 hidden units. Hindsight experience replay uses the "future" goal replay and selection strategy with $k = 4$ [48]. Here k controls the ratio of HER data to data coming from normal experience replay in the replay buffer. We use a similar reward function to that of SAVED, with a reward of -1 if the agent is not in the goal set and 0 if the agent is in the goal set. Additionally, for environments with constraints, we impose a reward of -100 when constraints are violated to encourage constraint satisfaction. The choice of collision reward is ablated in section 11.4.2. This reward is set to prioritize constraint satisfaction over task success, which is consistent with the selection of β in the model-based algorithms considered.

11.2 Simulated Experiments Additional Results

In Figure 11.1, we illustrate the mechanism by which SAVED iteratively improves upon suboptimal demonstrations on navigation task 2 by planning into an expanding safe set.

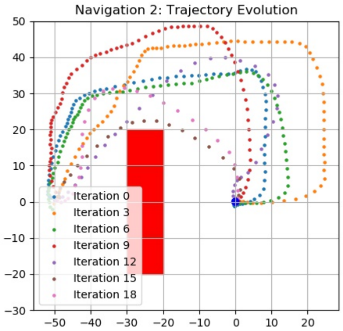


Figure 11.1: **Navigation Task 2 Trajectory Evolution:** SAVED rapidly improves upon demonstration trajectories by constraining its exploration to regions of relative certainty and high cost. By iteration 15, SAVED is able to find a safe but efficient trajectory to the goal at the origin.

In Figure 11.2, we show the task success rate for the PR2 reacher and Fetch pick and place tasks for SAVED and baselines. We note that SAVED outperforms RL baselines (except SAVED (No SS) for the reacher task, most likely because the task is relatively simple so the *sampled safe set* constraint has little effect) in the first 100 and 250 iterations for the reacher and pick and place tasks respectively. Note that although behavior cloning has a higher success rate, it does not improve upon demonstration performance. However, although SAVED's success rate is not as different from the baselines in these environments as those with constraints, this result shows that SAVED can be used effectively in a general purpose way, and still learns more efficiently than baselines in unconstrained environments as seen in the main section of this chapter.

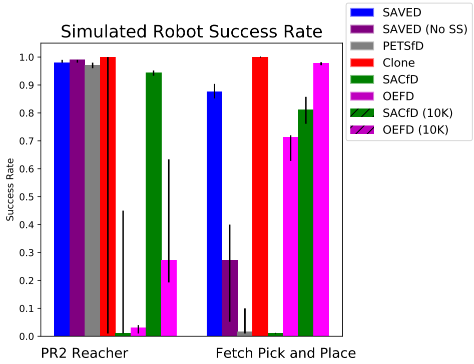


Figure 11.2: **Simulated Robot Experiments Success Rate:** SAVED has comparable success rate to Clone, PETSfD, and SAVED (No SS) on the reacher task in the first 100 iterations. For the pick and place task, SAVED outperforms all baselines in the first 250 iterations except for Clone, which does not improve upon demonstration performance.

11.3 Physical Experiments: Additional Details and Experiments

For all experiments, $\alpha = 0.05$ and a set of 100 hand-coded trajectories with a small amount of Gaussian noise added to the controls is generated. For all physical experiments, we use $\beta = 1$ for PETSfD since we found this gave the best performance when no signal from the value function was provided. In all tasks, the goal set is represented with a 1 cm ball in $\mathbb{R}^3$. The dVRK is controlled via delta-position control, with a maximum control magnitude set to 1 cm during learning for safety. We train state density estimator ρ on all prior successful trajectories for the physical robot experiments.

11.3.1 Figure-8

In addition to the knot-tying task discussed in the main section of this chapter, we also evaluate SAVED on a Figure-8 tracking task on the surgical robot. In this task, the dVRK must track a Figure 8 in the workspace. The agent is constrained to remain within a

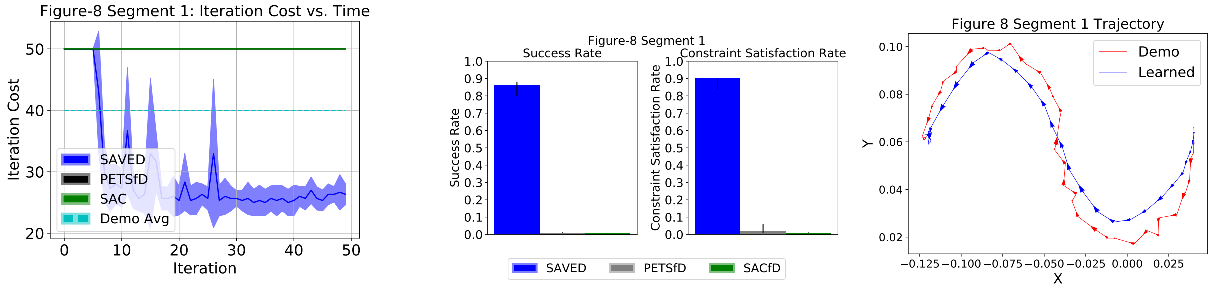


Figure 11.3: **Figure-8: Training Performance:** After just 10 iterations, SAVED consistently succeeds and converges to an iteration cost of 26, faster than demos which took an average of 40 steps. Neither baseline ever completes the task in the first 50 iterations; **Trajectories:** Demo trajectories satisfy constraints, but are noisy and inefficient. SAVED learns to speed up with only occasional constraint violations and stabilizes in the goal set.

1 cm pipe around a reference trajectory with chance constraint parameter $\beta = 0.8$ for SAVED and $\beta = 1$ for PETSfD. We use 100 inefficient but successful and constraint-satisfying demonstrations with average iteration cost of 40 steps for both segments. Additionally we use a planning horizon of 10 for SAVED and 30 for PETSfD. However, because there is an intersection in the middle of the desired trajectory, SAVED finds a shortcut to the goal state. Thus, the trajectory is divided into non-intersecting segments before SAVED separately optimizes each one. At execution-time, the segments are stitched together and we find that SAVED is robust enough to handle the uncertainty at the transition point. We hypothesize that this is because the dynamics and value function exhibit good generalization.

Results for both segments of the Figure 8 task are shown in Figures 11.3 and 11.4 below. In Figure 11.3, we see that SAVED quickly learns to smooth out demo trajectories while satisfying constraints, with a success rate of over 80% while baselines violate constraints on nearly every iteration and never complete the task, as shown in Figure 11.3. Note that PETSfD almost always violates constraints, even though constraints are enforced exactly as in SAVED. We hypothesize that since we need to give PETSfD a long planning horizon to make it possible to complete the task (since it has no value function), this makes it unlikely that a constraint satisfying trajectory is sampled with CEM. For the other segment of the Figure-8, SAVED still quickly learns to smooth out demo trajectories while satisfying constraints, with a success rate of over 80% while baselines violate constraints on nearly every iteration and never complete the task, as shown in Figure 11.4.

In Figure 11.5, we show the full trajectory for the Figure-8 task when both segments are combined at execution-time. This is done by rolling out the policy for the first segment, and then starting the policy for the second segment as soon as the policy for the first segment reaches the goal set. We see that even given uncertainty in the dynamics and end state for the first policy (it could end anywhere in a 1 cm ball around the goal position), SAVED is able to smoothly navigate these issues and interpolate between the two segments at execution-

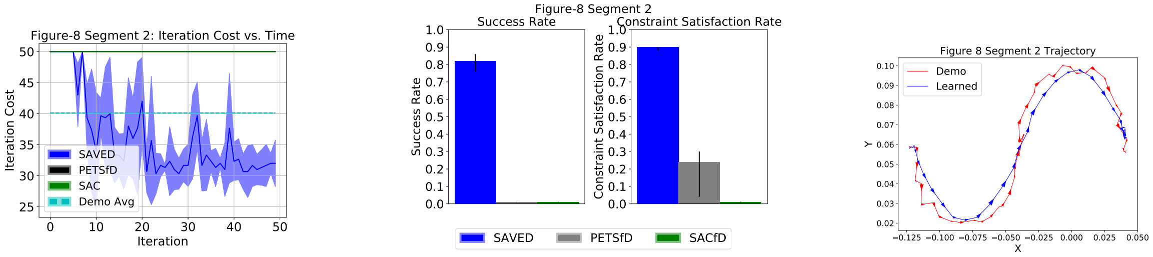


Figure 11.4: **Figure-8: Training Performance:** After 10 iterations, the agent consistently completes the task and converges to an iteration cost of around 32, faster than demos which took an average of 40 steps. Neither baseline ever completed the task in the first 50 iterations; **Trajectories:** Demo trajectories are constraint-satisfying, but noisy and inefficient. SAVED quickly learns to speed up demos with only occasional constraint violations and successfully stabilizes in the goal set. Note that due to the difficulty of the tube constraint, it is hard to avoid occasional constraint violations during learning, which are reflected by spikes in the iteration cost.

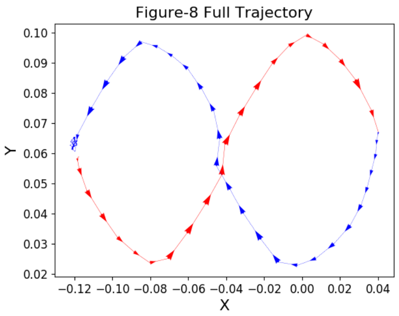


Figure 11.5: **Full Figure-8 trajectory:** We show the full figure-8 trajectory, obtained by evaluating learned policies for the first and second figure-8 segments in succession. Even when segmenting the task, the agent can smoothly interpolate between the segments, successfully navigating the uncertainty in the transition at execution-time and stabilizing in the goal set.

time to successfully stabilize at the goal at the end of the second segment. Each segment of the trajectory is shown in a different color for clarity. We suspect that SAVED's ability to handle this transition is reflective of good generalization of the learned dynamics and value functions.

11.3.2 Knot-Tying

In Figure 11.6, we show the full trajectory for both arms for the surgical knot-tying task. We see that the learned policy for arm 1 smoothly navigates around arm 2, after which arm 1 is manually moved down along with arm 2, which grasps the thread and pulls it up through the resulting loop in the thread, completing the knot.

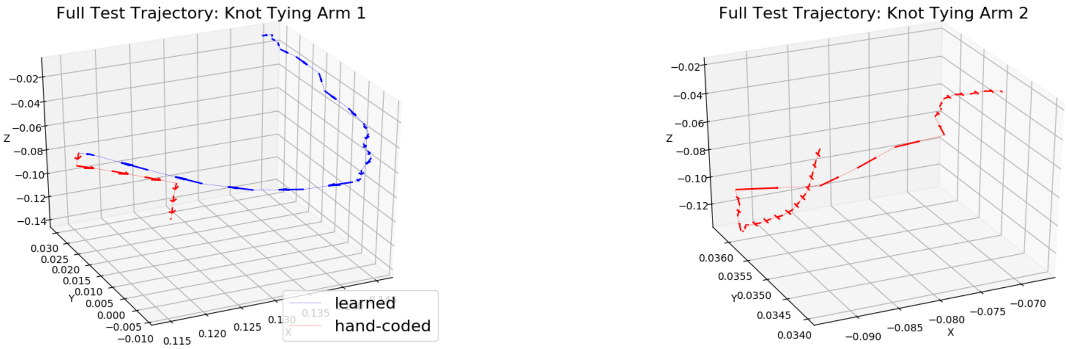


Figure 11.6: **Knot-Tying Full Trajectories: (a) Arm 1 trajectory:** We see that the learned part of the arm 1 trajectory is significantly smoothed compared to the demonstrations at execution-time as well, consistent with the training results. Then, in the hand-coded portion of the trajectory, arm 1 is simply moved down towards the phantom along with arm 2, which grasps the thread and pulls it up; **(b) Arm 2 trajectory:** This trajectory is hand-coded to move down towards the phantom after arm 1 has fully wrapped the thread around it, grasp the thread, and pull it up.

11.4 Ablations

11.4.1 SAVED

We investigate the impact of kernel width α, chance constraint parameter β, and the number of demonstrator trajectories used on navigation task 2. Results are shown in Figure 11.8. We see that SAVED is able to complete the task well even with just 20 demonstrations, but is more consistent with more demonstrations. We also notice that SAVED is relatively sensitive to the setting of kernel width α. When α is set too low, we see that SAVED is overly conservative, and thus can barely explore at all. This makes it difficult to discover regions near the goal set early on and leads to significant model mismatch, resulting in poor task performance. Setting α too low can also make it difficult for SAVED to plan to regions with high density further along the task, resulting in SAVED failing to make progress. On the other extreme, making α too large causes a lot of initial instability as the agent explores unsafe regions of the state space. Thus, α must be chosen such that SAVED is able to sufficiently explore, but does not explore so aggressively that it starts visiting states from which it has low confidence in being able reach the goal set. Reducing β allows the agent to take more risks, but this results in many more collisions. With $\beta = 0$, most rollouts end in collision or failure as expected. In the physical experiments, we find that allowing the agent to take some risk during exploration is useful due to the difficult tube constraints on the feasible state space.

Finally, we also ablate the quantity and quality of demonstrations used for navigation task 2 (Figure 11.8), and find that SAVED is still able to consistently complete the task with just 20 demonstrations and is relatively robust to lower quality demonstrations, although this does result in some instability during training. We additionally ablate the quantity and quality of demonstrations for navigation task 1 in Figure 11.7. We note that again, SAVED is relatively robust to varying demonstration quality, achieving similar performance even for very slow demonstrations

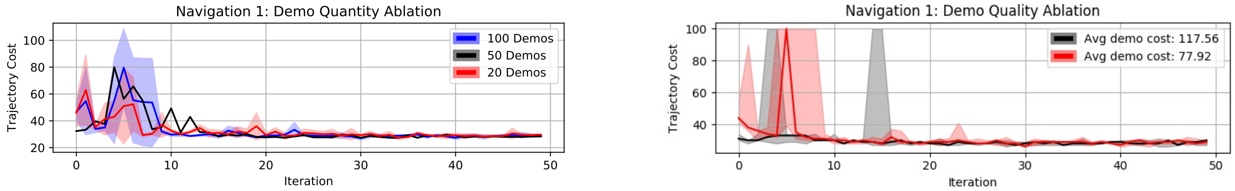


Figure 11.7: **SAVED Ablations on Navigation Task 1: Number of Demonstrations:** SAVED is able to consistently complete the task with just both demo qualities considered without significant performance decay. The 100 demonstrations provided in this task have average trajectory cost of 117.56 (black) and 77.82 (red) and SAVED significantly outperforms both, converging in less than 10 iterations in all runs to a policy with trajectory cost less than 30. **Demonstration quality:** SAVED is able to consistently complete the task with just 20 demonstrations (red) after 15 iterations. The demonstrations provided in this task have average trajectory cost of 77.82.

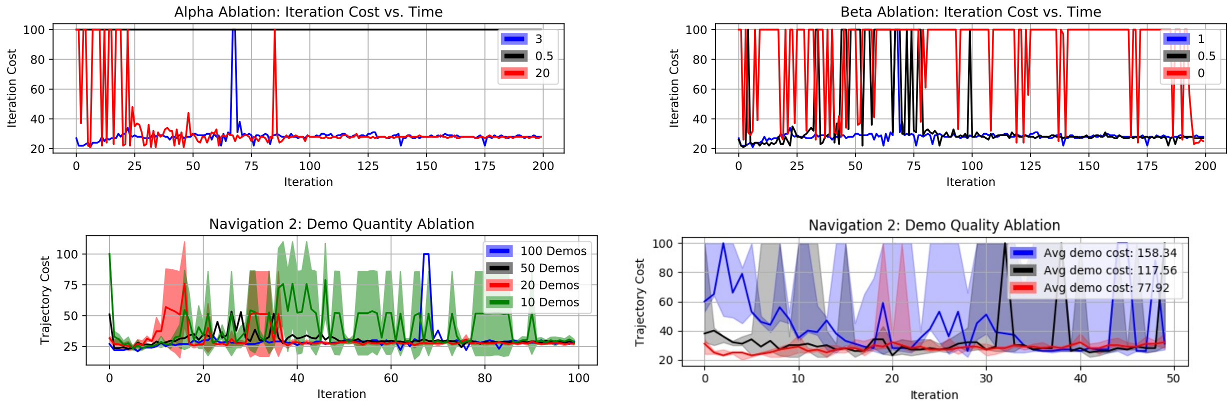


Figure 11.8: **SAVED Ablations on Navigation Task 2: Kernel width α:** We see that α must be chosen to be high enough such that SAVED is able to explore enough to find the goal set, but not so high that SAVED starts to explore unsafe regions of the state space; **Chance constraint parameter β:** Decreasing β results in many more collisions with the obstacle. Ignoring the obstacle entirely results in the majority of trials ending in collision or failure. **Demonstration quantity:** In this experiment, we vary the number of demonstrations that SAVED is provided. We see that SAVED is able to complete the task with just 20 demonstrations (red), but more demonstrations result in increased stability during learning. Even with 10 demonstrations (green), SAVED is able to sometimes complete the task. The demonstrations provided in this task have average trajectory cost of 77.82. **Demonstration quality:** SAVED efficiently learns a controller in all runs in all cases, the worst of which has demos that attain an iteration cost 5 times higher than the converged controller. We do occasionally observe some instability in the value function, which begins to display somewhat volatile behavior after initially finding a good controller. Constraints are never violated during learning in any of the runs.

11.4.2 Model-Free

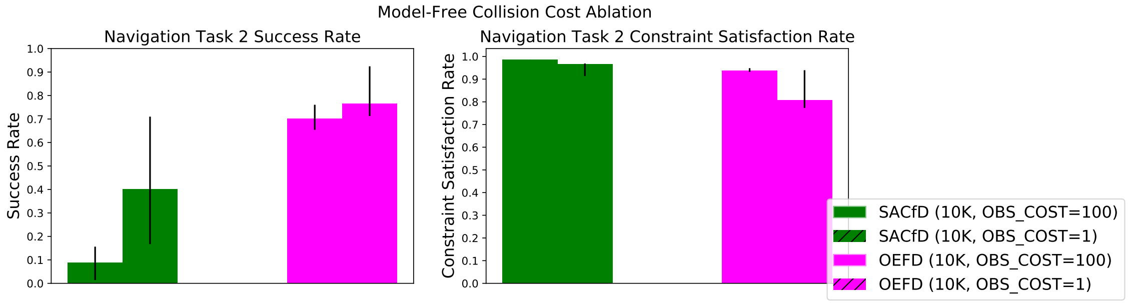


Figure 11.9: A high cost for constraint violations results in conservative behavior that learns to avoid the obstacle, but also makes it take longer to learn to perform the task. Setting the cost low results in riskier behavior that more often achieves task success.

To convey information about constraints to model-free methods, we provide an additional cost for constraint violations. We ablate this parameter for navigation task 2 in Figure 11.9. We find that a high cost for constraint violations results in conservative behavior that learns to avoid the obstacle, but also takes much longer to achieve task success. Setting the cost low results in riskier behavior that succeeds more often. This trade-off is also present for model-based methods, as seen in the prior ablations. Additionally, we also ablate the demonstration quality for the model-free baselines, and find that increasing the iteration cost of the demonstrations by almost 50% does not significantly change the learning curve of OEFD (Figure 11.10), and in both cases, OEFD takes much longer than SAVED to start performing the task successfully (Figure 11.7). We also perform the same study on the model-free RL baseline algorithm Soft Actor Critic from Demonstrations (SACfD) [62]. We observe that increasing demonstration length results in somewhat faster learning, and hypothesize this could be due to the replay buffer having more data to initially train from. We note that this method has high variance across the runs, and all runs took close to 900 iterations to converge (Figure 11.10) while SAVED converges in less than 10 iterations (Figure 11.7). We also ablate demo quantity for SACfD on navigation task 1 in Figure 11.11 and find that although SACfD has a performance improvement with additional demonstrations, it takes a few hundred iterations to converge and more than a 100 iterations to even complete the task, while SAVED converges within 15 iterations (Figure 11.7).

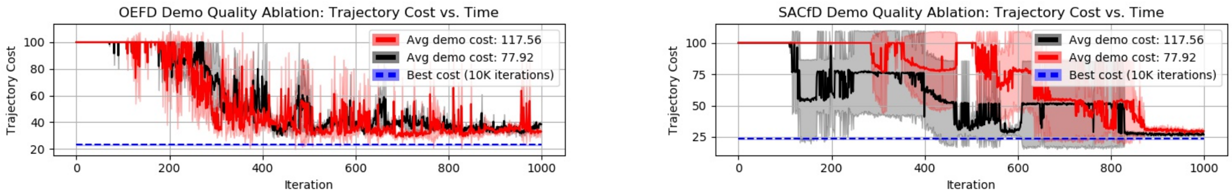


Figure 11.10: **SAVED Model-Free RL Demo Quality Ablations on Navigation Task 1: OEFD:** We see that the baseline OEFD has similar performance across demonstration qualities. OEFD takes hundreds of iterations to start performing the task successfully, while SAVED converges less than 10 iterations; **SACfD:** We see that the baseline SACfD does slightly better with worse demonstrations. This could be due to the fact that more samples are placed in the agent's replay buffer with longer demonstrations. We note that both cases take hundreds of iterations to start completing the task, while SAVED starts to the complete the task almost immediately.

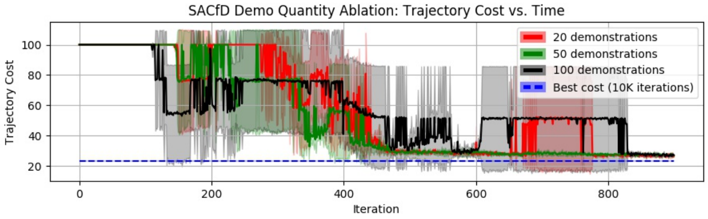


Figure 11.11: **SACfD Demo Quantity Ablation on Navigation Task 1:** We study the effect of varying demonstration numbers on the model free RL baseline algorithm SACfD [62]. We see that the baseline SACfD has high variance across all demonstration quantities, and takes roughly similar time to converge in all settings with 50 demonstrations (green) being the fastest. We also plot the best observed cost in 10,000 iterations across all runs (dashed blue) and note that unlike OEFD (Figure 11.10), the SACfD runs all converge close to this value.

Chapter 12

Appendix for Chapter 6

In Sections 12.0.1 and 12.0.2 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 Section 12.0.3. In Section 12.0.4, we present an additional experiment studying the task success rate of LS^3 and comparisons evolves as training progresses. Finally, in Section 12.0.5 we perform sensitivity experiments and ablations.

12.0.1 Algorithm Details

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

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. [82], 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

$$J(\theta) = \|f_{\text{dec}}(z_t) - s_t\|_2^2 + \beta D_{KL}\left(f_{\text{enc}}(z_t|s_t)||\mathcal{N}(0,1)\right) \tag{12.1}$$

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

Probabilistic Dynamics: As in Chua et al. [107] 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:

$$J(\theta) = -\log f_{\text{dyn},\theta}(z_{t+1}|z_t, a_t) \tag{12.2}$$

When using f_{dyn} for planning, we use the TS-1 method from Chua et al. [107].

Value Functions: As discussed in Section 4.4.3, we train an ensemble of recursively defined value functions to predict long term reward. We represent these functions using fully connected neural networks with 3 hidden layers with 256 hidden units. Similarly to [24], we use separate training objectives during offline and online training. During offline training, we train the function to predict actual discounted cost-to-go on all trajectories in $\mathcal{D}$. Hence, for a latent vector z_t, the loss during offline training is given as follows where V has parameter θ:

$$J(\theta) = \left(V_\theta^\pi(z_t) - \sum_{i=1}^{T-t} \gamma^i r_{t+i}\right)^2 \tag{12.3}$$

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

$$J(\theta) = \left(V_\theta^\pi(z_t) - (r_t + \gamma V_{\theta'}^{\pi'}(z_{t+1}))\right)^2 \tag{12.4}$$

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

$$f_{\mathbb{S}}(s_t) = \max(\mathbb{1}_{\mathbb{S}^j}(s_t), \gamma_{\mathbb{S}} f_{\mathbb{S}}(s_{t+1})) \tag{12.5}$$

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 [24, 62, 238], 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 4.2. We build on the implementation of the cross entropy method provided in [234], 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 4.2 where constraints are implemented by assigning large negative rewards to trajectories which violate either the safe set constraint or user-specified constraints. This process is repeated for $n_{\text{cem\_iters}}$ to iteratively refine the set of sampled trajectories to optimize equation 4.2. To improve the optimizer's efficiency on tasks where subsequent actions are often correlated, we sample a proportion $(1 - p_{\text{random}})$ of the optimizer's candidates at the first iteration from the distribution it learned when planning the last action. To avoid local minima, we sample a proportion p_{random} uniformly from the action space. See Chua et al. [107] for more details on the cross entropy method as applied to planning over neural network dynamics models.

As mentioned in Section 4.4.4, we set $\delta_{\mathbb{S}}$ for the safe set classifier $f_{\mathcal{S}}$ adaptively by checking whether there exists at least one plan which satisfies the safe set constraint at each CEM iteration. If no such plan exists, we multiply $\delta_{\mathbb{S}}$ by 0.8 and re-initialize the optimizer at the first CEM iteration with the new value of $\delta_{\mathbb{S}}$. We initialize $\delta_{\mathbb{S}} = 0.8$.

Soft Actor-Critic from Demonstrations (SACfD)

We utilize the implementation of the Soft Actor Critic algorithm from [236] and initialize the actor and critic from demonstrations but keep all other hyperparameters the same as the default in the provided implementation. We create a new dataset $\mathcal{D}_{\text{demos}} \subsetneq \mathcal{D}$ using only data from the suboptimal demonstrator, and use the data from $\mathcal{D}_{\text{demos}}$ to behavior clone the actor and initialize the critic using offline bellman backups. We use the same mean-squared loss function for behavior cloning as for the behavior clone policy, but only train the mean of the SAC policy. Precisely, we use the following loss for some policy π with parameter θ: $L(\theta, \mathcal{D}_{\text{demos}}) = \sum_{\tau_i \in \mathcal{D}_{\text{demos}}} \sum_{t=1}^{T} ||\mu_\theta(s_t^i) - a_t^i||^2$ where s_t^i and a_t^i are the state and action at timestep t of trajectory τ_i and $\pi(\cdot|s_t) \sim \mathcal{N}(\mu_\theta(s_t), \sigma_\phi(s_t))$. We also experimented with training the SAC critic on all data provided to LS^3 in $\mathcal{D}$ but found that this hurt performance.

We use the architecture from [236] 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.

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

We build on the implementation of the Recovery RL algorithm [8] provided in [239]. 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 $(\gamma_{\text{risk}} = 0.55, \epsilon_{\text{risk}} = 0.7)$.

Advantage Weighted Actor-Critic (AWAC)

To provide a comparison to state of the art offline reinforcement learning algorithms, we evaluate AWAC [109] on the experimental domains in this chapter. We use the implementation of AWAC from [240]. 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 [240] for all other hyperparameters.

12.0.2 LS^3 Implementation Details

In Table 12.1, 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

Table 12.1: 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

use the Adam optimizer with a learning rate of 1×10^{-4}, except for dynamics which use a learning rate of 1×10^{-3}.

12.0.3 Experimental Domain Details

Navigation

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

Reacher

The reacher domain is built on the reacher domain provided in the DeepMind Control Suite from [110]. The robot is represented with a planar 2-link arm and the agent supplies torques to each of the 2 joints. Because velocity is not observable from a single frame, algorithms are provided with several stacked frames as input. The start position of the end-effector is fixed and the objective is to navigate the end effector to a fixed goal set on the top left of the workspace without allowing the end effector to enter a large red stay-out zone. To collect data from $\mathcal{D}_{\text{constraint}}$ we randomly sample starting states in the environment, and then use a PID controller to move towards the constraint. To sample random data that will require the agent to model velocity for accurate prediction, we start trajectories at random places in the environment, and then sample each action from a normal distribution centered around the previous action, $a_{t+1} \sim \mathcal{N}(a_t, \sigma^2 I)$ for $\sigma^2 = 0.2$. We collect 50 demonstrations of successful behavior, 50 trajectories containing constraint violations and 100 short (length 20) trajectories or random data.

Sequential Pushing

This sequential pushing environment is implemented in MuJoCo [229], and the robot can specify a desired planar displacement $a = (\Delta x, \Delta y)$ for the end effector position. The goal is to push all 3 blocks backwards by at least some displacement on the table, but constraints are violated if blocks are pushed backwards off of the table. For the sequential pushing environment, demonstrations are created by guiding the end effector to the center of each block and then moving the end effector in a straight line at a low velocity until the block is in the goal set. This same process is repeated for each of the 3 blocks. Data of constraint violations and random transitions for $\mathcal{D}_{\text{constraint}}$ and $\mathcal{D}_{\text{rand}}$ are collected by randomly switching between a policy that moves towards the blocks and a policy that randomly samples from the action space. We collect 500 demonstrations of successful behavior and 300 trajectories of random and/or constraint violating behavior.

Physical Cable Routing

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

12.0.4 Additional Results

We additionally study how the task success rate of LS^3 and comparisons evolves as training progresses in Figure 12.1. 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.

12.0.5 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 12.2 and 12.3. 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.

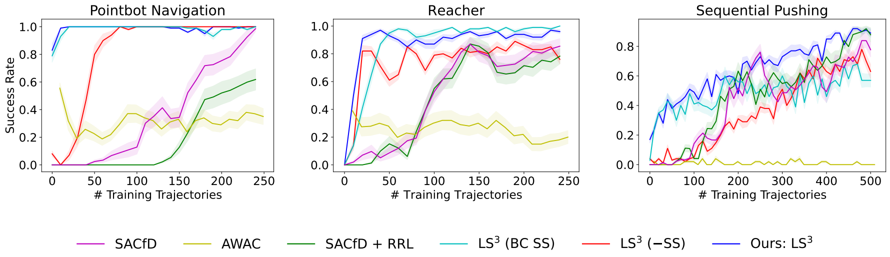


Figure 12.1: **Task Success Rate:** Learning curves showing mean and standard error of task success rate of checkpointed policies over 10 random seeds (and 10 rollouts per seed). We see that LS^3 has a much higher task success rate than comparisons early on, and maintains a success rate at least as high as comparisons throughout training in all environments.

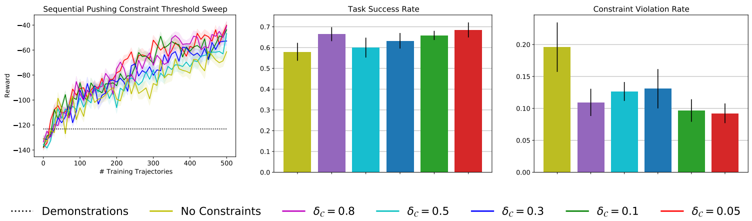


Figure 12.2: **Hyperparameter Sweep for LS^3 Constraint Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathcal{C}}$ on the sequential pushing environment. As expected, we see that without avoiding latent space obstacles (No Constraints) the agent violates constraints more often, while lower thresholds (meaning the planning algorithm is more conservative) generally lead to fewer violations.

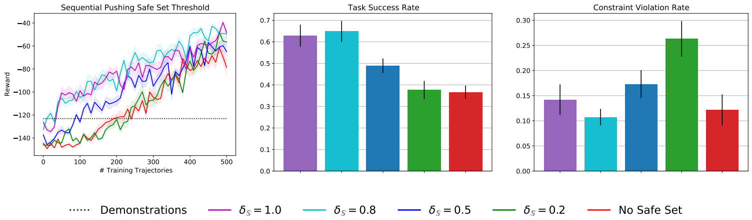


Figure 12.3: **Hyperparameter Sweep for LS^3 Safe Set Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathbb{S}}$ on the sequential pushing environment. We see that after offline training, the agent can successfully complete the 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.

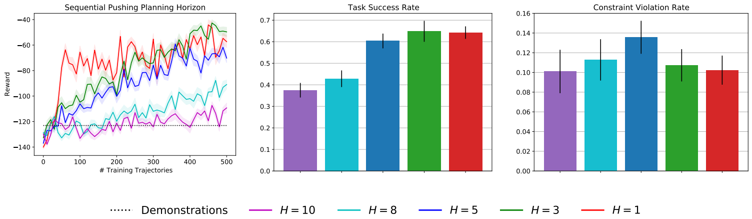


Figure 12.4: **Hyperparameter Sweep for LS^3 Planning Horizon:** Plots show mean and standard error over 10 random seeds for experiments with different settings of H on the sequential pushing environment. We see that when the planning horizon is too high the agent cannot reliably complete the task due 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.

Chapter 13

Appendix for Chapter 7

13.1 Static Regret

13.1.1 Proof of Chapter 5.3.1

Recall the standard notion of static regret as defined in Definition 5.3.1:

$$\text{Regret}_N^S((\psi_i)_{i=1}^N) = \sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_i) - l_i(\pi_{\theta^*}, \psi_i)] \text{ where } \theta^* = \arg\min_{\theta \in \Theta} \sum_{i=1}^N l_i(\pi_\theta, \psi_i) \tag{13.1}$$

However, we seek to bound

$$\text{Regret}_N^S(\psi_N) = \sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta^\star}, \psi_N)] \text{ where } \theta^\star = \arg\min_{\theta \in \Theta} \sum_{i=1}^N l_i(\pi_\theta, \psi_N) \tag{13.2}$$

as defined in Definition 5.3.2.

Notice that this corresponds to the static regret of the agent with respect to the losses parameterized by the last observed supervisor ψ_N. We can do this as follows:

$$\text{Regret}_N^S(\psi_N) = \sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta^\star}, \psi_N)] \tag{13.3}$$

$$= \sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta^\star}, \psi_N)] - \text{Regret}_N^S((\psi_i)_{i=1}^N) + \text{Regret}_N^S((\psi_i)_{i=1}^N) \tag{13.4}$$

$$\begin{aligned} &= \sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)] + \sum_{i=1}^N [l_i(\pi_{\theta^*}, \psi_i) - l_i(\pi_{\theta^\star}, \psi_N)] \\ &\quad + \text{Regret}_N^S((\psi_i)_{i=1}^N) \end{aligned} \tag{13.5}$$

$$\begin{aligned} &\leq \sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)] + \sum_{i=1}^N [l_i(\pi_{\theta^\star}, \psi_i) - l_i(\pi_{\theta^\star}, \psi_N)] \\ &\quad + \text{Regret}_N^S((\psi_i)_{i=1}^N) \end{aligned} \tag{13.6}$$

Here, inequality 13.6 follows from the fact that $\sum_{i=1}^N l_i(\pi_{\theta^*}, \psi_i) \leq \sum_{i=1}^N l_i(\pi_{\theta^\star}, \psi_i)$. Now, we can focus on bounding the extra term. Let $h(x, y) = \|x - y\|^2$.

$$\sum_{i=1}^N [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)] + \sum_{i=1}^N [l_i(\pi_{\theta^\star}, \psi_i) - l_i(\pi_{\theta^\star}, \psi_N)] \tag{13.7}$$

$$\begin{aligned} &= \sum_{i=1}^N \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^T h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)) - h(\pi_{\theta_i}(s_t^i), \psi_i(s_t^i)) \right] \\ &\quad + \sum_{i=1}^N \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^T h(\pi_{\theta^\star}(s_t^i), \psi_i(s_t^i)) - h(\pi_{\theta^\star}(s_t^i), \psi_N(s_t^i)) \right] \end{aligned} \tag{13.8}$$

$$\begin{aligned} &\leq \sum_{i=1}^N \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^T \langle \nabla_\psi h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)), \psi_N(s_t^i) - \psi_i(s_t^i) \rangle \right] \\ &\quad + \sum_{i=1}^N \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^T \langle \nabla_\psi h(\pi_{\theta^\star}(s_t^i), \psi_i(s_t^i)), \psi_i(s_t^i) - \psi_N(s_t^i) \rangle \right] \end{aligned} \tag{13.9}$$

$$
\begin{aligned}
&= \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle 2(\psi_N(s_t^i) - \pi_{\theta_i}(s_t)), \psi_N(s_t^i) - \psi_i(s_t^i) \rangle \right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle 2(\psi_i(s_t^i) - \pi_{\theta^\star}(s_t)), \psi_i(s_t^i) - \psi_N(s_t^i) \rangle \right]
\end{aligned}
\tag{13.10}
$$

$$
\begin{aligned}
&\leq \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2\|\psi_N(s_t^i) - \pi_{\theta_i}(s_t)\| \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2\|\psi_i(s_t^i) - \pi_{\theta^\star}(s_t)\| \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right]
\end{aligned}
\tag{13.11}
$$

$$
\leq 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] \tag{13.12}
$$

Equation 13.8 follows from applying the definition of the loss function. Inequality 13.9 follows from applying convexity of h in ψ. Equation 13.10 follows from evaluating the corresponding gradients. Inequality 13.11 follows from Cauchy-Schwarz and inequality 13.12 follows from the action space bound. Thus, we have:

$$
\text{Regret}_N^S(\psi_N) \leq 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] + \text{Regret}_N^S((\psi_i)_{i=1}^N) \tag{13.13}
$$

13.1.2 Proof of Chapter 5.3.1

$$
\forall s \in \mathcal{S},\ \forall N > i, \|\psi_i(s) - \psi_N(s)\| \leq f_i \text{ where } \lim_{i \to \infty} f_i = 0 \tag{13.14}
$$

implies that

$$
\mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] \leq f_i \ \forall N > i \in \mathbb{N} \tag{13.15}
$$

This in turn implies that

$$
\sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] \leq \sum_{i=1}^{N} f_i \tag{13.16}
$$

Remark: For sublinearity, we really only need inequality 13.15 to hold. Due to the dependence of $p(\tau|\theta_i)$ on the parameter θ_i of the policy at iteration i, we tighten this assumption with the stricter Cauchy condition 13.14 to remove the dependence of a component of the regret on the sequence of policies used.

The Additive Cesàro's Theorem states that if the sequence $(a_n)_{n=1}^{\infty}$ has a limit, then

$$\lim_{n\to\infty} \frac{a_1 + a_2 \dots a_n}{n} = \lim_{n\to\infty} a_n$$

Thus, we see that if $\lim_{i\to\infty} f_i = 0$, then it must be the case that $\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^N f_i = 0$. This shows that for some $(f_i)_{i=1}^N$ converging to 0, it must be the case that

$$\sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^T \|\psi_i(s_t^i) - \psi_N(s_t^i)\|\right] \leq \sum_{i=1}^N f_i = o(N)$$

Thus, based on the regret bound in Chapter 5.3.1, we can achieve sublinear $\text{Regret}_N^S(\psi_N)$ for any sequence $(f_i)_{i=1}^N$ which converges to 0 given an algorithm that achieves sublinear $\text{Regret}_N^S((\psi_i)_{i=1}^N)$:

$$\text{Regret}_N^S(\psi_N) = \text{Regret}_N^S((\psi_i)_{i=1}^N) + o(N)$$

□

13.2 Dynamic Regret

13.2.1 Proof of Chapter 5.3.1

Recall the standard notion of dynamic regret as defined in Definition 5.3.3:

$$\text{Regret}_N^D((\psi_i)_{i=1}^N) = \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_i) - l_i(\pi_{\theta_i^*}, \psi_i)\right] \text{ where } \theta_i^* = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_i) \tag{13.17}$$

However, we seek to bound

$$\text{Regret}_N^D(\psi_N) = \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i^\star}, \psi_N)\right] \text{ where } \theta_i^\star = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_N) \tag{13.18}$$

as defined in Definition 5.3.4.

Notice that this corresponds to the dynamic regret of the agent with respect to the losses parameterized by the most recent supervisor ψ_N. We can do this as follows:

$$\begin{aligned}
\text{Regret}_N^D(\psi_N) &= \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i^\star}, \psi_N)\right] && (13.19)\\
&= \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i^\star}, \psi_N)\right] - \text{Regret}_N^D((\psi_i)_{i=1}^N) && (13.20)\\
&\quad + \text{Regret}_N^D((\psi_i)_{i=1}^N) \\
&= \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)\right] + \sum_{i=1}^N \left[l_i(\pi_{\theta_i^*}, \psi_i) - l_i(\pi_{\theta_i^\star}, \psi_N)\right] && (13.21)\\
&\quad + \text{Regret}_N^D((\psi_i)_{i=1}^N) \\
&\leq \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)\right] + \sum_{i=1}^N \left[l_i(\pi_{\theta_i^\star}, \psi_i) - l_i(\pi_{\theta_i^\star}, \psi_N)\right] && (13.22)\\
&\quad + \text{Regret}_N^D((\psi_i)_{i=1}^N)
\end{aligned}$$

Here, inequality 13.22 follows from the fact that $l_i(\pi_{\theta_i^*}, \psi_i) \leq l_i(\pi_{\theta_i^\star}, \psi_i)$. Now as before, we can focus on bounding the extra term. Let $h(x, y) = \|x - y\|^2$.

$$\begin{aligned}
&\sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)\right] + \sum_{i=1}^N \left[l_i(\pi_{\theta_i^\star}, \psi_i) - l_i(\pi_{\theta_i^\star}, \psi_N)\right] && (13.23)\\
&= \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)) - h(\pi_{\theta_i}(s_t^i), \psi_i(s_t^i))\right] && (13.24)\\
&\quad + \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T h(\pi_{\theta_i^\star}(s_t^i), \psi_i(s_t^i)) - h(\pi_{\theta_i^\star}(s_t^i), \psi_N(s_t^i))\right] \\
&\leq \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T \langle \nabla_\psi h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)), \psi_N(s_t^i) - \psi_i(s_t^i)\rangle\right] && (13.25)\\
&\quad + \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T \langle \nabla_\psi h(\pi_{\theta_i^\star}(s_t^i), \psi_i(s_t^i)), \psi_i(s_t^i) - \psi_N(s_t^i)\rangle\right]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle 2(\psi_N(s_t^i) - \pi_{\theta_i}(s_t)), \psi_N(s_t^i) - \psi_i(s_t^i) \rangle \right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle 2(\psi_i(s_t^i) - \pi_{\theta_i^\star}(s_t)), \psi_i(s_t^i) - \psi_N(s_t^i) \rangle \right] \qquad (13.26) \\
&\le \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2\|\psi_N(s_t^i) - \pi_{\theta_i}(s_t)\| \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2\|\psi_i(s_t^i) - \pi_{\theta_i^\star}(s_t)\| \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] \qquad (13.27) \\
&\le 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] \qquad (13.28)
\end{aligned}
$$

The steps of this proof follow as in the proof of the static regret reduction. Equation 13.24 follows from applying the definition of the loss function. Inequality 13.25 follows from applying convexity of h in ψ. Equation 13.26 follows from evaluating the corresponding gradients. Inequality 13.27 follows from Cauchy-Schwarz and inequality 13.28 follows from the action space bound. Combining this bound with 13.22, we have our desired result:

$$
\text{Regret}_N^D(\psi_N) \le 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] + \text{Regret}_N^D((\psi_i)_{i=1}^N) \qquad (13.29)
$$

□

13.2.2 Proof of Chapter 5.3.2

By Chapter 5.3.1,

$$
\sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] = o(N)
$$

which implies that

$$
\text{Regret}_N^D(\psi_N) = \text{Regret}_N^D((\psi_i)_{i=1}^N) + o(N)
$$

□

13.2.3 Predictability of Online Learning Problems

Next, we establish that the online learning problem defined by the losses defined in Section 5.2 is an (α, β)-predictable online learning problem as defined in Cheng et al. [129]. An online learning problem is (α, β)-predictable if it satisfies $\forall \theta \in \Theta$, (1) $l_i(.)$ is α strongly convex in θ,

(2) $\|\nabla_\theta l_{i+1}(\pi_\theta, \psi_{i+1}) - \nabla_\theta l_i(\pi_\theta, \psi_i)\| \leq \beta\|\theta_{i+1} - \theta_i\| + \zeta_i$ where $\sum_{i=1}^N \zeta_i = o(N)$. Proposition 12 in Cheng et al. [129] shows that for (α, β)-predictable problems, sublinear dynamic regret can be achieved if $\alpha > \beta$. Furthermore, Theorem 3 in Cheng et al. [129] shows that if α is sufficiently large and β sufficiently small, then sublinear dynamic regret can be achieved by online gradient descent.

Lemma 13.2.1. *If $\forall s \in \mathcal{S}$, $\forall N > i$, $\|\psi_i(s) - \psi_N(s)\| \leq f_i$ where $\lim_{i\to\infty} f_i = 0$, the learning problem is $(\alpha, 4G\eta \sup_{a\in\mathcal{A}} \|a\|)$-predictable in θ: $l_i(\pi_\theta, \psi)$ is α-strongly convex by assumption and if Assumption 5.3.1 holds, then $l_i(\pi_\theta, \psi)$ satisfies:*

$$\|\nabla_\theta l_{i+1}(\pi_\theta, \psi_{i+1}) - \nabla_\theta l_i(\pi_\theta, \psi_i)\| \leq 4G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| + \zeta_i \textit{ where } \sum_{i=1}^N \zeta_i = o(N)$$

Proof of Chapter 13.2.1 We have bounded $\text{Regret}_N^D(\psi_N)$ by the sum of $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ and a sublinear term. Now, we analyze $\text{Regret}_N^D((\psi_i)_{i=1}^N)$. We note that we can achieve sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ if the losses satisfy

$$\|\nabla_\theta l_{i+1}(\pi_\theta, \psi_{i+1}) - \nabla_\theta l_i(\pi_\theta, \psi_i)\| \leq \beta\|\theta_{i+1} - \theta_i\| + \zeta_i$$

where $\sum_{i=1}^N \zeta_i = o(N)$ by Proposition 12 in Cheng et al. [129].

Note that for $J_\tau(\pi_\theta, \psi) = \frac{1}{T}\sum_{t=1}^T \|\psi(s_t) - \pi_\theta(s_t)\|^2$, we have

$$\begin{aligned}
\nabla_\theta l_i(\pi_\theta, \psi) &= \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \frac{1}{T}\sum_{t=1}^T \nabla_\theta \|\psi(s_t) - \pi_\theta(s_t)\|^2 && (13.30)\\
&= \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \nabla_\theta J_\tau(\pi_\theta, \psi) && (13.31)\\
&= \int p(\tau|\theta_i)\nabla_\theta J_\tau(\pi_\theta, \psi) d\tau && (13.32)\\
\nabla_\theta J_\tau(\pi_\theta, \psi) &= \frac{1}{T}\sum_{s_t\in\tau} 2\nabla_\theta\pi_\theta(s_t)^T(\pi_\theta(s_t) - \psi(s_t)) && (13.33)\\
&= \frac{2}{T}\nabla_\theta\pi_\theta(\tau)^T(\pi_\theta(\tau) - \psi(\tau)) && (13.34)
\end{aligned}$$

where

$$\psi(\tau) = \begin{bmatrix}\psi(s_0)\\ \vdots \\ \psi(s_T)\end{bmatrix},\ \pi_\theta(\tau) = \begin{bmatrix}\pi_\theta(s_0)\\ \vdots \\ \pi_\theta(s_T)\end{bmatrix},\ \nabla_\theta\pi_\theta(\tau) = \begin{bmatrix}\nabla_\theta\pi_\theta(s_0)\\ \vdots \\ \nabla_\theta\pi_\theta(s_T)\end{bmatrix} \tag{13.35}$$

Taking the difference of the above loss gradients, we obtain:

$$\|\nabla_\theta l_{i+1}(\pi_\theta, \psi_{i+1}) - \nabla_\theta l_i(\pi_\theta, \psi_i)\| \tag{13.36}$$

$$= \left\| \int p(\tau|\theta_{i+1}) \nabla_\theta J_\tau(\pi_\theta, \psi_{i+1}) d\tau - \int p(\tau|\theta_i) \nabla_\theta J_\tau(\pi_\theta, \psi_i) d\tau \right\| \tag{13.37}$$

$$\leq \int \|p(\tau|\theta_{i+1}) \nabla_\theta J_\tau(\pi_\theta, \psi_{i+1}) - p(\tau|\theta_i) \nabla_\theta J_\tau(\pi_\theta, \psi_i)\| d\tau \tag{13.38}$$

$$\begin{aligned} = \int \bigg\| &\frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T (p(\tau|\theta_i)\psi_i(\tau) - p(\tau|\theta_{i+1})\psi_{i+1}(\tau)) \\ &+ \frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T (p(\tau|\theta_{i+1})\pi_\theta(\tau) - p(\tau|\theta_i)\pi_\theta(\tau)) \bigg\| d\tau \end{aligned} \tag{13.39}$$

$$\begin{aligned} \leq &\int \left\| \frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T (p(\tau|\theta_i)\psi_i(\tau) - p(\tau|\theta_{i+1})\psi_{i+1}(\tau)) \right\| d\tau \\ &+ \int \left\| \frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T \pi_\theta(\tau) (p(\tau|\theta_{i+1}) - p(\tau|\theta_i)) \right\| d\tau \end{aligned} \tag{13.40}$$

$$\begin{aligned} \leq &\int \left\| \frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T (p(\tau|\theta_i)\psi_i(\tau) - p(\tau|\theta_{i+1})\psi_{i+1}(\tau)) \right\| d\tau \\ &+ 2G \sup_{a \in \mathcal{A}} \|a\| \int |p(\tau|\theta_{i+1}) - p(\tau|\theta_i)| d\tau \end{aligned} \tag{13.41}$$

$$\leq \int \left\| \frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T (p(\tau|\theta_i)\psi_i(\tau) - p(\tau|\theta_{i+1})\psi_{i+1}(\tau)) \right\| d\tau + 2G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \tag{13.42}$$

$$\leq \frac{2}{T} G \int \|p(\tau|\theta_i)\psi_i(\tau) - p(\tau|\theta_{i+1})\psi_{i+1}(\tau)\| d\tau + 2G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \tag{13.43}$$

$$\begin{aligned} = &\frac{2}{T} G \int \|p(\tau|\theta_i)\psi_i(\tau) - p(\tau|\theta_i)\psi_{i+1}(\tau) + p(\tau|\theta_i)\psi_{i+1}(\tau) - p(\tau|\theta_{i+1})\psi_{i+1}(\tau)\| d\tau \\ &+ 2G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \end{aligned} \tag{13.44}$$

$$\begin{aligned} \leq &\frac{2}{T} G \int \|p(\tau|\theta_i)(\psi_i(\tau) - \psi_{i+1}(\tau))\| + \|(p(\tau|\theta_i) - p(\tau|\theta_{i+1}))\psi_{i+1}(\tau)\| d\tau \\ &+ 2G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \end{aligned} \tag{13.45}$$

$$\leq \frac{2}{T} G \int p(\tau|\theta_i) \|\psi_i(\tau) - \psi_{i+1}(\tau)\| d\tau + 4G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \tag{13.46}$$

$$\leq 2Gf_i \int p(\tau|\theta_i) d\tau + 4G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \tag{13.47}$$

$$\leq 2Gf_i + 4G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| \tag{13.48}$$

$$= 4G\eta \sup_{a \in \mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| + \zeta_i \tag{13.49}$$

where here $\zeta_i = 2Gf_i$ and we see that $2G\sum_{i=1}^{N} f_i = o(N)$ as desired for some $(f_i)_{i=1}^{N}$ where $\lim_{i\to\infty} f_i = 0$ as in Chapter 5.3.1. Equation 13.37 follows from applying definitions. Equation 13.38 follows from the triangle inequality. Equation 13.39 follows from substitution of the loss gradients. Inequality 13.40 follows from the triangle inequality and factoring out common terms. Inequality 13.41 follows from subadditivity, the policy Jacobian and action space bound. Inequality 13.42 follows from Assumption 5.3.1. Equation 13.43 follows from subadditivity of the operator norm and the policy Jacobian bound. Equation 13.45 follows from the triangle inequality, and equation 13.46 follows from the triangle inequality and Assumption 5.3.1. Equations 13.47 and 13.49 follow from the convergence assumption of the supervisor and the triangle inequality. □

Lemma 13.2.2. *Assumption 5.2.2 implies that the loss function gradients are bounded as follows:*

$$\|\nabla_\theta l_i(\pi_\theta, \psi)\| \leq 2G\delta \quad \forall \theta, \theta_i \in \Theta,\ \forall \psi$$

Proof of Chapter 13.2.2

$$\left\| \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2(\nabla_\theta \pi_\theta(s_t^i))^T \left(\pi_\theta(s_t^i) - \psi_i(s_t^i)\right) \right] \right\| \leq$$

$$\mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \left\| 2(\nabla_\theta \pi_\theta(s_t^i))^T \left(\pi_\theta(s_t^i) - \psi_i(s_t^i)\right) \right\| \right]$$

by convexity of norms $\|\cdot\|$ and Jensen's inequality.

Then, we have that

$$\|(\nabla_\theta \pi_\theta(s))^T(\pi_\theta(s) - \psi(s))\| \leq \|\nabla_\theta \pi_\theta(s)\| \|\pi_\theta(s) - \psi(s)\| \leq G\delta \ \ \forall \theta \in \Theta,\ \forall s \in \mathcal{S},\ \forall \psi$$

due to subadditivity and the assumption that the action space diameter is bounded. Thus, we have that

$$\forall \theta, \theta_i \in \Theta, \forall \psi, \|\nabla_\theta l_i(\pi_\theta, \psi)\| \leq 2G\delta \quad \square$$

13.2.4 Proof of Chapter 5.3.2

From Chapter 13.2.1, the loss gradients are bounded by the sum of a Lipschitz-type term and a sublinear term, satisfying the conditions for Proposition 12 from Cheng et al. [129]. Thus, by Proposition 12 from Cheng et al. [129], we see that as long as $\alpha > 4G\eta \sup_{a\in\mathcal{A}}\|a\|$, there exists an algorithm that can achieve sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$. An example of an algorithm that achieves sublinear dynamic regret under this condition is the greedy algorithm [129]: $\theta_{i+1} = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_i)$.

Define $\beta = 4G\eta \sup_{a\in\mathcal{A}}\|a\|$, $\lambda = \beta/\alpha$, and $\xi_i = \zeta_i/\alpha$. For the greedy algorithm, the result can be shown in a similar fashion to Theorem 3 of Cheng et al. [129]:

$$\|\theta_i^* - \theta_i\| = \|\theta_i^* - \theta_{i-1}^*\| \leq \lambda\|\theta_i - \theta_{i-1}\| + \frac{\zeta_i}{\alpha} \leq \lambda^i\|\theta_1 - \theta_0\| + \sum_{j=1}^{i} \lambda^{i-j}\xi_j$$

where the first inequality follows from Proposition 1 of Lee et al. [124] and the second inequality follows from repeated application of the same proposition. Summing from 1 to N with $\zeta_i = 2Gf_i$ as in the proof of Chapter 5.3.2, we have

$$\sum_{i=1}^{N}\sum_{j=1}^{i} \lambda^{i-j}\xi_j \leq \sum_{i=1}^{N} \xi_i(1+\lambda+\lambda^2+\ldots) \leq \frac{1}{1-\lambda}\sum_{i=1}^{N}\xi_i = \frac{2G}{\alpha(1-\lambda)}\sum_{i=1}^{N} f_i$$

Thus, if $\sum_{i=1}^{N} f_i = o(N)$, we can show that the greedy algorithm achieves sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ by using the Lipschitz continuity of the losses as shown in the proof of Chapter 13.2.2 if the parameter space diameter is bounded as follows: $D = \sup_{\theta,\theta' \in \Theta} \|\theta - \theta'\|$.

$$\begin{aligned}
\text{Regret}_N^D((\psi_i)_{i=1}^N) &\leq 2G\delta \sum_{i=1}^{N} \|\theta_i - \theta_i^*\| \\
&\leq 2G\delta\left(D\sum_{i=1}^{N}\lambda^i + \frac{2G}{\alpha(1-\lambda)}\sum_{i=1}^{N} f_i\right) \\
&\leq 2G\delta\left(\frac{D}{1-\lambda} + \frac{2G}{\alpha(1-\lambda)}\sum_{i=1}^{N} f_i\right) \\
&= o(N)
\end{aligned}$$

For the last part of the lemma, the fact that online gradient descent achieves sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ follows directly from applying Theorem 3 from Cheng et al. [129] with $\frac{4G\eta \sup_{a\in\mathcal{A}}\|a\|}{\alpha} > \frac{\alpha}{2\gamma}$ if the losses are γ-smooth in θ. □

13.2.5 Proof of Chapter 5.3.2

The proof follows immediately from combining the result of Chapter 5.3.2 and Chapter 5.3.2. □

13.3 Training Details

13.3.1 CSF Learner

For the linear policy, the CSF learner is trained via linear regression with regularization parameter $\alpha = 1$. For the neural network policy, the CSF learner is represented with an ensemble of 5 neural networks, each with 2 layers with 20 hidden units and swish activations.

13.3.2 PETS

PETS learns an ensemble of neural network dynamics models using sampled transitions and updates them on-policy to better reflect the dynamics local to the learned policy's state distribution. We use the implementation from [234]. MPC is run over the learned dynamics to select actions for the next iteration. For all environments, a probabilistic ensemble of 5 neural networks with 3 hidden layers, each with 200 hidden units and swish activations are used to represent the dynamics model. The TS-∞ sampling procedure is used for planning. We use an MPC planning horizon of length 25 for all environments and 1 initial random rollout to seed the learned dynamics model. Chua et al. [25] contains further details on training PETS.

13.3.3 SAC

We use the rlkit implementation [236] of soft actor critic with the following parameters: batch size = 128, discount factor = 0.99, soft target $\tau = 0.001$, policy learning rate = 0.0003, Q function learning rate = 0.0003, value function learning rate = 0.0003, and replay buffer size = 1000000. All networks are two-layer multi-layer perceptrons with 300 hidden units.

13.3.4 TD3

We use the rlkit implementation [236] of TD3 with the following parameters: batch size = 128, discount factor = 0.99, and replay buffer size = 1000000. The exploration strategy consists of adding Gaussian noise $\mathcal{N}(0, 0.1)$ to actions chosen by the policy. All networks are two-layer multi-layer perceptrons with 300 hidden units.

13.3.5 ME-TRPO

We model both the policy and dynamics with neural networks, using an ensemble of dynamics models to avoid exploitation of model bias. We use the ME-TRPO implementation from [241] with the following hyperparameters: batch size=128, discount factor=1, and learning rate =.001 for both the policy and dynamics. The policy network has two hidden layers with 64 units each and all dynamics networks have two hidden layers with 512 units each and ReLU activation.

13.4 Experimental Details

13.4.1 Simulated Experiments

Both simulated experiments involve manipulation tasks on a simulated PR2 robot and are from the provided code in Chua et al. [25]. Both are implemented as 7-DOF torque control tasks. For all tasks, we plot the sum of rewards for each training episode.

13.4.2 Physical Experiments

Both physical experiments involve delta-position control in 3D space on the daVinci surgical system, which is cable driven and hard to precisely control, making it difficult to reliably reach a desired pose without appropriate compensation [60]. The CSF learner policy and supervisor dynamics are modeled by 3 hidden-layer feed-forward neural networks with 200 hidden units each. The tasks involve guiding the end effectors to targets in the workspace and isotropic concave quadratic rewards are used. For all tasks, we plot the sum of rewards for each training episode. For multi-arm experiments, the arms are limited to subsets of the state space where collisions are not possible. We are investigating modeling arm collisions for future work. Since the da Vinci surgical system has relatively limited control frequency, although the CSF learner often enables significantly faster query time than PETS, the improvement in policy evaluation time was somewhat less significant due to physical hardware constraints. In future work, we plan to implement the proposed algorithm on a robot with higher frequency control capability.

Chapter 14

Appendix for Chapter 8

The supplementary material is structured as follows: In Section 14.1 we discuss brief theoretical motivation for Recovery RL and possible variants and in Section 14.2 we discuss algorithmic details for Recovery RL and comparison algorithms. In Section 14.3, we report additional metrics for all domains and comparisons and in Section 14.4 we visualize the safety critic for all navigation experiments. We provide additional details about algorithm implementation in Section 14.5, and on domain-specific algorithm hyperparameters in Section 14.6. Finally, we report simulation and physical environment details in Section 14.7.

14.1 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.

14.1.1 Theoretical Motivation

Recall that the task policy is operating in an environment with modified dynamics:

$$P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a) = \begin{cases} P(s'|s,a) & (s,a) \in \mathcal{T}^{\pi}_{\text{safe}} \\ P(s'|s,a^{\pi_{\text{rec}}}) & (s,a) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{14.1}$$

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}.

14.1.2 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.

14.1.3 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.

14.2 Algorithm Details

14.2.1 Recovery RL

Recovery Policy: In principle, any off-policy reinforcement learning algorithm can be used to learn the recovery policy π_{rec}. In this chapter, 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 [135]. 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:

$$L_\theta(s_t, a_t) = \mathbb{E}\left[\sum_{i=0}^{H} \hat{Q}^{\pi}_{\phi,\text{risk}}(\hat{s}_{t+i}, a_{t+i})\right] \tag{14.2}$$

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. [25] 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 $\hat{Q}^{\pi}_{\phi,\text{risk}}$ 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 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 [62] 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 $\hat{Q}^{\pi}_{\phi,\text{risk}}$ 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.

14.2.2 Unconstrained

We use an implementation of the popular model-free reinforcement learning algorithm Soft Actor Critic [62, 236], which maximizes a combination of task reward and policy entropy with a stochastic actor function.

14.2.3 Lagrangian Relaxation (LR)

In this section we will briefly motivate and derive the Lagrangian relaxation comparison. 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.

14.2.4 Risk Sensitive Policy Optimization (RSPO)

We implement Risk Sensitive Policy Optimization by implementing the Lagrangian Relaxation method as discussed in Section 14.2.3 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 [140].

14.2.5 Safety Q-Functions for Reinforcement Learning (SQRL)

This comparison 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 $Q^{\pi}_{\text{risk}}(s_t, a_t) \leq \epsilon_{\text{risk}}$.

14.2.6 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.

14.2.7 Off Policy Reward Constrained Policy Optimization (RCPO)

In on-policy RCPO [3], the policy is optimized via policy gradient estimators by maximizing $\mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\left(\gamma^t R(s,a) - \lambda\gamma^t_{\text{risk}} D(s,a)\right)\right]$. In this chapter, we use $D(s,a) = Q^{\pi}_{\text{risk}}(s,a)$ 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 comparison. 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.

14.3 Additional Experimental Metrics

In Figure 14.1 and Figure 14.2, we report cumulative task successes and constraint violations for all methods for all simulation experiments. We report these statistics for the image-based obstacle avoidance physical experiment in Figure 14.4. 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.

Additionally, in Figures 14.3 and 14.5, we plot the cumulative reward attained by the agent for Recovery RL and all comparison algorithms to evaluate whether Recovery RL learns more efficiently than comparisons while also learning safely. For all plots, we show total reward attained in each episode smoothed over a 100 episode length window. Additionally, we do not show results when constraints are violated to prevent negative bias for algorithms which may violate constraints very frequently, which accounts for gaps in the plot, especially for the unconstrained algorithm which tends to violate constraints very frequently. Thus, the plots illustrate the attained reward for all algorithms conditioned on not violating constraints, which provides a good measure on the quality of solutions found. We find that in addition to the safe learning shown by Recovery RL as evidenced by the results in Figure 14.2, the results in Figure 14.3 and Figure 14.5 indicate that when Recovery RL satisfies constraints, it generally converges to higher quality solutions more quickly compared to the comparison algorithms. These results provide further evidence that the way Recovery RL separates the often conflicting objectives of task directed optimization and constraint satisfaction allows it to not only be safer during learning, but also learn more efficiently.

In Table 14.1, we report empirical results for when constraint violations occur in Table 14.1. Results suggest that in most tasks, the recovery policy is already activated when violations do occur. Thus, in these failure cases, Recovery RL is able to successfully predict

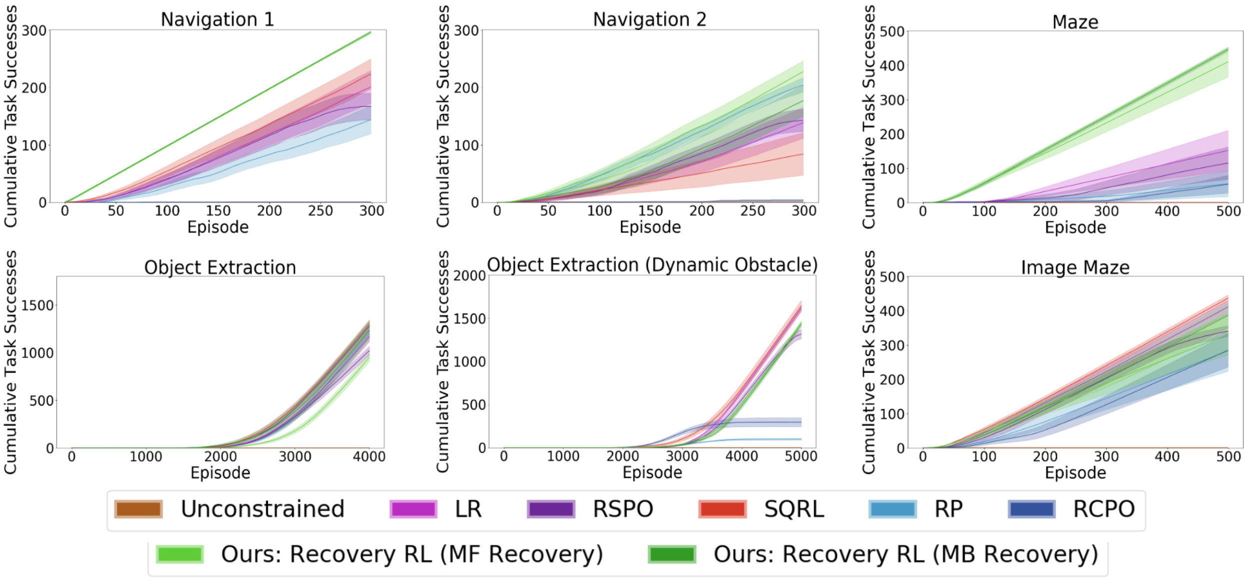


Figure 14.1: **Simulation Experiments Cumulative Successes:** We plot the cumulative task successes for each algorithm in each simulation domain, with results averaged over 10 runs for all algorithms. We observe that Recovery RL (green), is generally among the most successful algorithms. In the cases that it has lower successes, we observe that it is safer (Figure 14.2). 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.

future violations, but is not able to prevent them. This is encouraging, and suggests that for environments in which a recovery policy is very challenging to learn, Recovery RL could still be used to query a human supervisor for interventions.

14.4 Safety Critic Visualizations

We visualize the safety critic after pretraining for the navigation domains in Figure 14.7 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.

14.5 Implementation Details

Here we overview implementation and hyperparameter details for Recovery RL and all comparisons. 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,

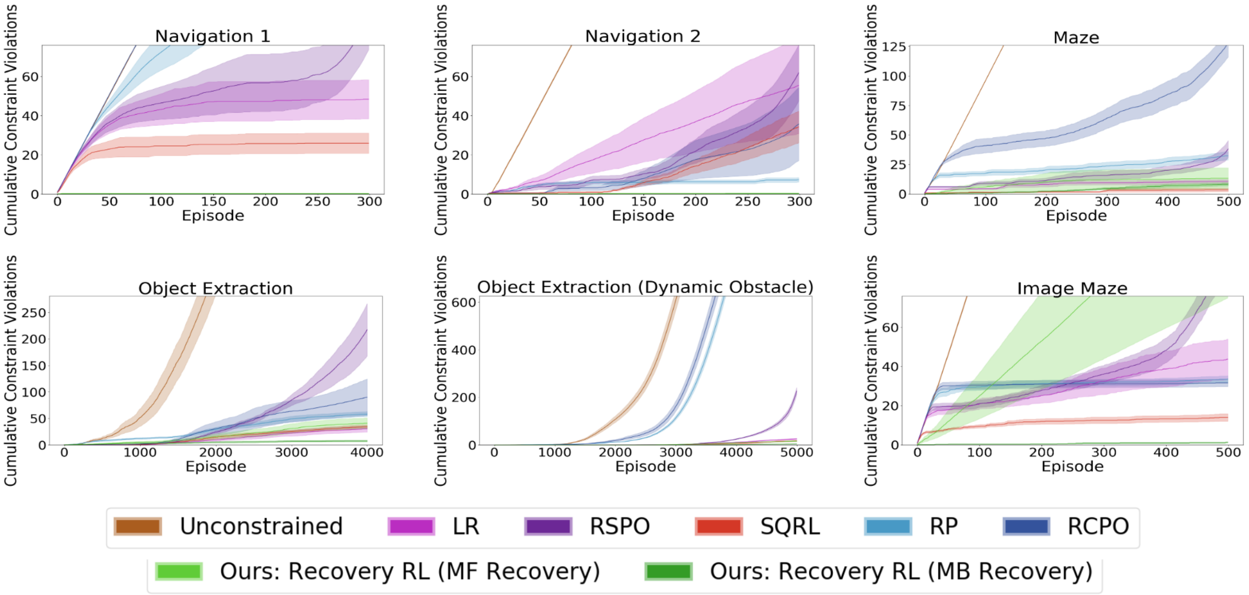


Figure 14.2: **Simulation Experiments Cumulative Violations:** We plot the cumulative constraint violations for each algorithm in each simulation domain, with results averaged over 10 runs for all algorithms. We observe that Recovery RL (green), is among the safest algorithms across all domains. In the cases where it is less safe than a comparison, it has a higher task success rate (Figure 14.1).

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 [89] provided in [236] 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 comparisons in Section 14.5.2 and then discuss the implementation of the recovery policy for Recovery RL in Section 14.5.3.

14.5.1 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

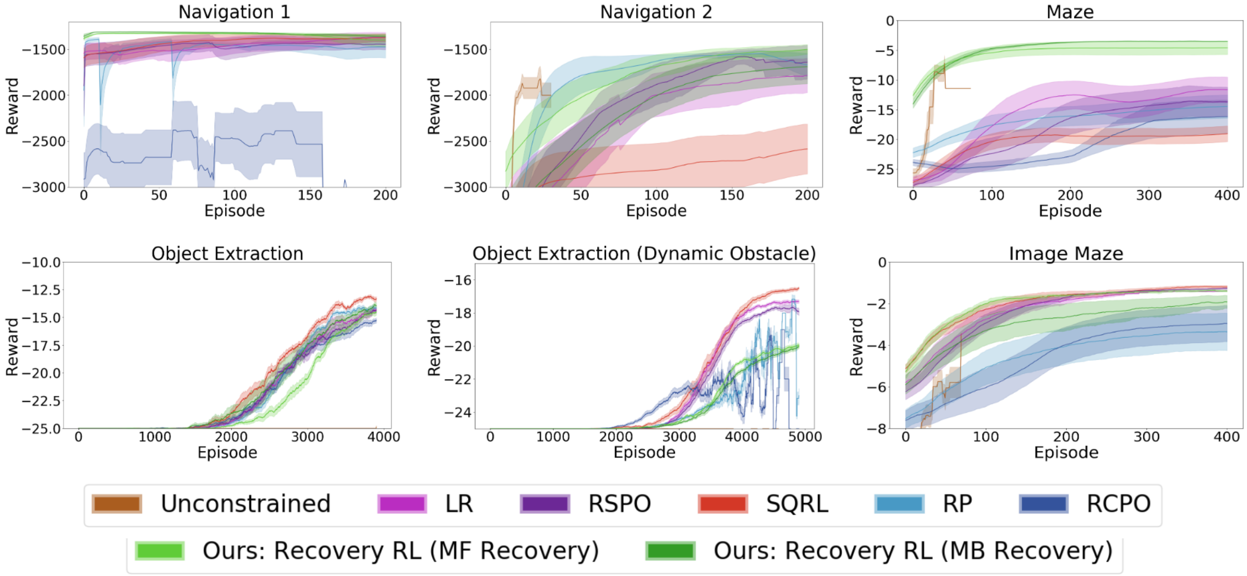


Figure 14.3: **Simulation Experiments Reward Learning Curve:** We show the total reward attained in each episode smoothed over a 100 episode length window for each simulation domain, with results averaged over 10 runs for all algorithms. We do not show results when constraints are violated to prevent negative bias for algorithms which may violate constraints very frequently, which accounts for gaps in the plot, especially for the unconstrained algorithm which tends to violate constraints very frequently. Thus, the plots illustrate the attained reward for all algorithms conditioned on not violating constraints, which provides a good measure on the quality of solutions found. We find that on all but the dynamic obstacle domain, Recovery RL is able to converge more quickly to higher quality solutions with respect to the task reward function compared to comparisons. This indicates that Recovery RL is able to learn more efficiently, in addition to more safely, compared to comparison algorithms.

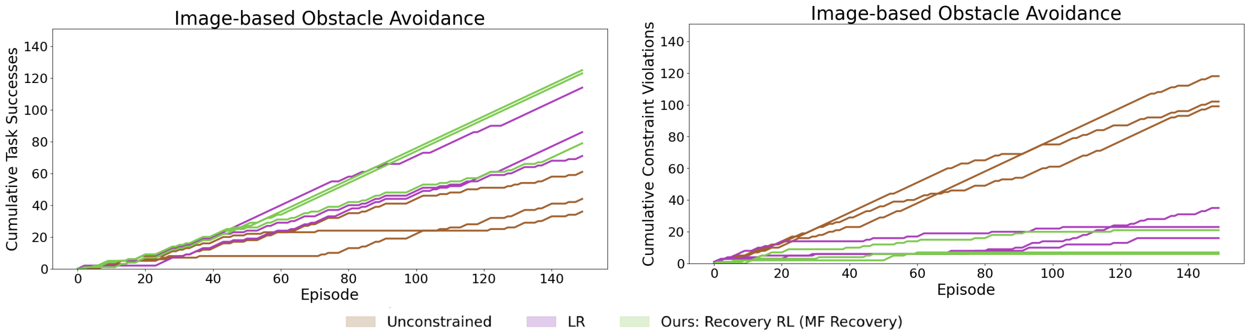


Figure 14.4: **Physical Experiment Successes and Violations:** We plot the cumulative constraint violations and task successes for the image-based obstacle avoidance task on the dVRK for all 3 runs of each algorithm. We observe that Recovery RL is generally both more successful and safer than LR and unconstrained.

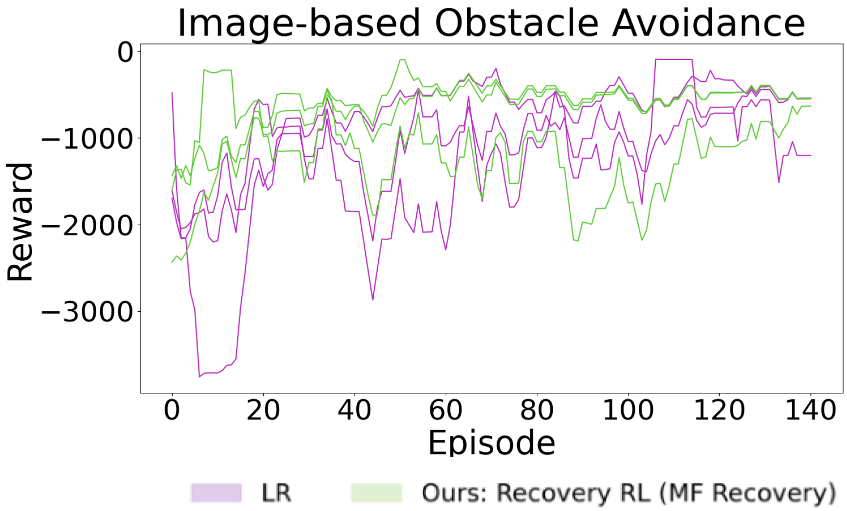


Figure 14.5: **Physical Experiment Reward Learning Curve:** We show the total reward attained in each episode smoothed over a 10 episode length window with results from 3 runs for all algorithms. We do not show results when constraints are violated to prevent negative bias for algorithms which may violate constraints very frequently, which accounts for gaps in the plot or explains why some plots have a single line (only one out of 3 runs is constraint satisfying) . Note that the Unconstrained algorithm does not appear in the plot as it never makes progress on the harder initial configuration of the task. Thus, the plots illustrate the attained reward for all algorithms conditioned on not violating constraints, which provides a good measure on the quality of solutions found. We find that Recovery RL is able to converge more quickly to higher quality solutions with respect to the task reward function compared to comparisons. This indicates that Recovery RL is able to learn more efficiently, in addition to more safely, compared to comparison algorithms.

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.

14.5.2 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

Table 14.1: **Constraint Violations Breakdown:** We report the proportion of constraint violations for each environment that occur when the recovery policy is activated for Recovery RL (format is mean ± standard error). If most constraint violations occur when the recovery policy is active, this indicates that the safety critic is sufficiently accurate to detect that the recovery policy must be activated, but may not provide sufficient information to avoid constraint violations. We note that if the safety critic detects the need for recovery behavior too late, then these errors are attributed to the recovery policy. For the both Maze environments and the Object Extraction environment, most constraint violations occur when the recovery policy is activated. In Navigation 1, none occur when the recovery policy is activated, but in this environment constraints are almost never violated. In the Image-Based obstacle avoidance tasks, most violations occur when the recovery policy is not activated, which indicates that the bottleneck in this task 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 physical robot. In the Dynamic Obstacle Object Extraction environment, we observe a more even combination of low safety critic values and recovery errors during constraint violations.

	Navigation 1	**Navigation 2**	**Maze**	**Object Extraction**	**Object Extraction (Dynamic Obstacle)**	**Image Maze**	**Image Obstacle Avoidance**
MF Recovery	N/A	N/A	0.828 ± 0.115	0.954 ± 0.024	0.550 ± 0.049	0.717 ± 0.156	0.000 ± 0.000
MB Recovery	N/A	1.000 ± 0.000	0.858 ± 0.039	0.98344 ± 0.01655	0.269 ± 0.055	0.583 ± 0.059	N/A

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.

14.5.3 Recovery Policy Training Details

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

Model-Free Recovery

The model-free recovery policy uses the same architecture as the task policy for all tasks, as described in Section 14.5.1. 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 [62]. The recovery policy is trained at each timestep on a batch of 1000 samples from the replay buffer.

Model-Based Recovery Training Details

For the non-image-based model-based recovery policy, we use PETS [25, 242], 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. [25] 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. [86]. 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. [86]. 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 deconvolutional 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.

14.6 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 Tables 14.3 and 14.2.

14.7 Environment Details

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

Table 14.2: Hyperparameters for Recovery RL and comparisons for all domains

	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	$(0.65, 0.2, 1000)$	3000	$(0.65, 0.2, 5000)$	$(0.65, 0.2)$	$(0.65, 0.2, 5)$
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 Obstacle Avoidance	$(0.55, 0.05, 1000)$	N/A	N/A	$(0.55, 0.05)$	N/A

Table 14.3: Hyperparameters for Recovery RL and all comparisons.

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	$(\gamma_{\text{risk}}, \epsilon_{\text{risk}}, H)$

14.7.1 Navigation Environments

The Navigation 1 and 2 environments have linear Gaussian dynamics and are built from scratch while the Maze environment is built on the Maze environment from [86]. In all navigation environments, offline data is collected by repeatedly initializing the pointmass agent randomly in the environment and executing controls to make it collide with the nearest obstacle.

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 offline pretraining. For Navigation 1, 455 of these transitions contain constraint violating states, while in Navigation 2, 778 of these transitions contain constraint violating states.

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 offline pretraining, 1163 of which contain constraint violating states.

14.7.2 Manipulation Environments

We build two manipulation environments on top of the cartgripper environment in the visual foresight repository [81]. The robot can translate in cardinal directions and open/close its grippers. In manipulation environments, offline data is collected by tuning a proportional controller to guide the robot end effector towards the objects. For the offline constraint violations, Gaussian noise is added to the controls when the end effector is sufficiently close to the objects to increase the likelihood of constraint violations. Additionally, to seed the task critic function for SAC to ease exploration for all algorithms, we utilize the same PID controller to collect task demos illustrating the red object being successfully lifted by automatically opening and closing the gripper when the end effector is sufficiently close to the red object.

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 offline pretraining, 363 of which contain constraint violating states. 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 offline pretraining, 896 of which contain constraint violating states. All methods are given 1000 transitions of task demonstration data to pretrain the task policy's critic function.

14.7.3 Image Maze

This maze is also implemented in MuJoCo with different walls from the maze that has ground-truth state. 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, 3466 of which contain constraint violating states.

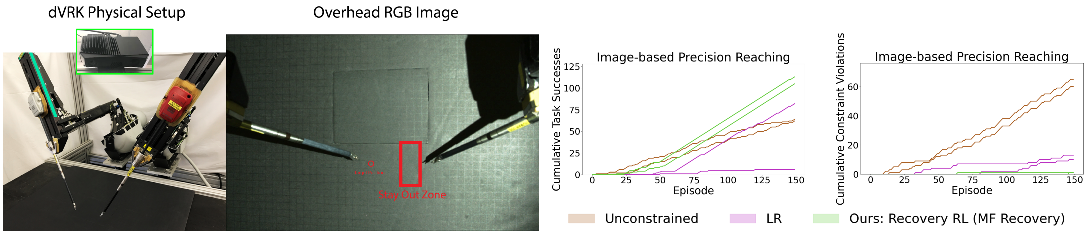


Figure 14.6: **Additional Physical Experiment:** The image reacher task on the dVRK involves guiding the end effector to a target position while avoiding an invisible stay out zone in the center of the workspace. We plot the cumulative constraint violations and task successes the image reacher task on the dVRK. We observe that Recovery RL is both more successful and safer than LR and unconstrained.

14.7.4 Physical Experiments

Physical experiments are run on the da Vinci Research Kit (dVRK) [78], 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, and we subsample the images so input images have dimensions 48 × 64 × 3. End effector position is checked *by the environment* using the robot's odometry to check 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 [151], but we ignore these effects in this chapter. We train a neural network to classify whether the robot is in collision based on its current readings and joint position. All methods that use a safety critic are supplied with 6, 000 transitions of data for pretraining, 649 of which contain constraint violating states. As for the navigation environments, offline data is collected by randomly initializing the end effector in the environment and guiding it towards the nearest obstacle. To reduce extrapolation errors during learning, we sample a start state on the right and left sides of the workspace with equal probability.

14.7.5 Additional Physical Experiment

We also evaluate Recovery RL and comparisons on an image-based reaching task where the robot must make sure the end effector position does not intersect with a stay out zone in the center of the workspace instead of physical bumpers. The setup is almost identical to the setup described in Section 14.7.4. We again provide RGB images to algorithms, and use 10,000 transitions to pre-train the safety critic. We again find that Recovery RL is able to outperform comparisons on this task, both in terms of constraint satisfaction, and task completion.

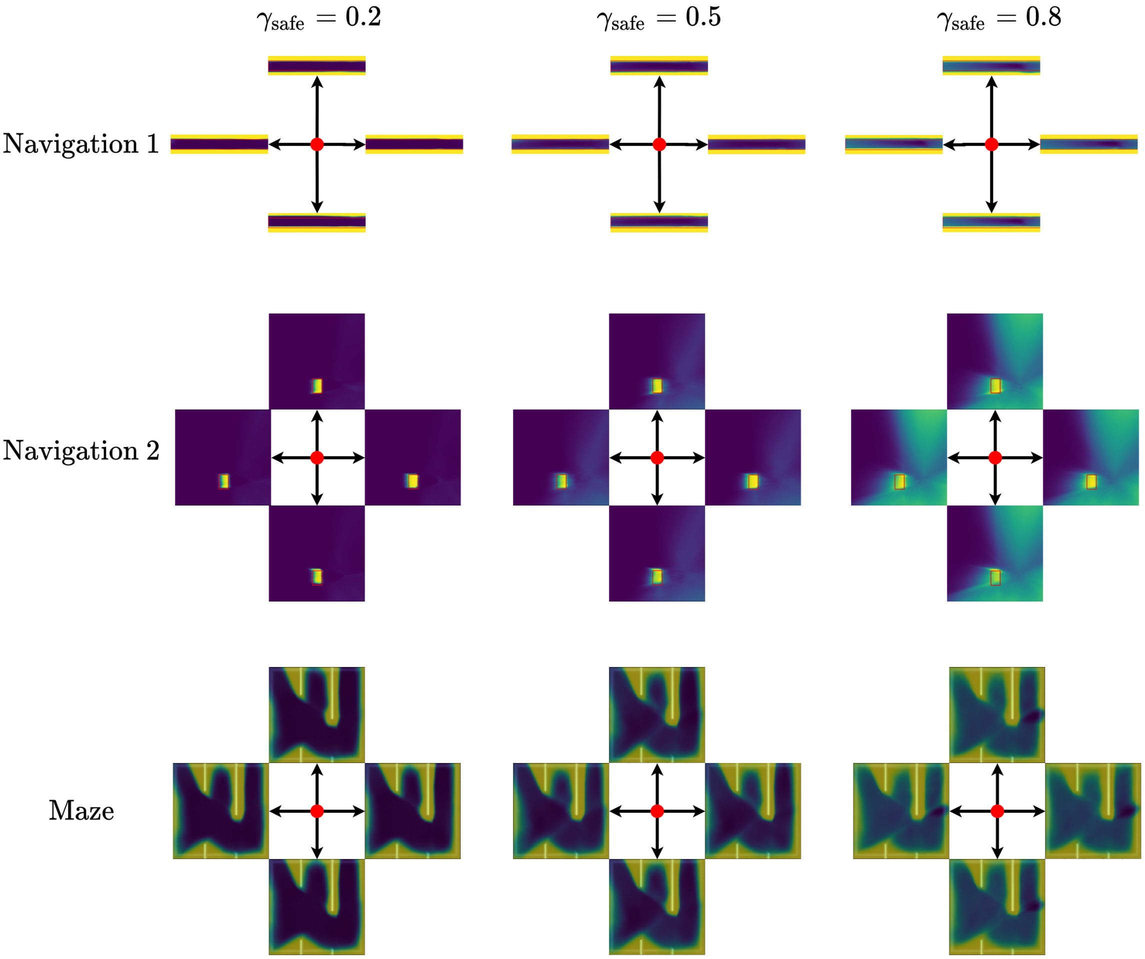


Figure 14.7: $\hat{Q}^{\pi}_{\phi,\text{risk}}$ **Visualization:** We plot the safety critic Q^{π}_{risk} for the navigation environments using the cardinal directions (left, right, up, down) as action input. We see that as γ_{risk} is increased, the gradient is lower, and the the function more gradually 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.

Chapter 15

Appendix for Chapter 9

15.0.1 Algorithm Description

The full detail of the MESA algorithm is described in Algorithm 5. In Phase 1, Offline Meta-Learning, the safety critic is updated with a MAML-style objective. In Phase 2, both the safety critic and recovery policy adapt to the test environment with a small offline test dataset. Finally, in Phase 3, the agent interacts with the test environment by using Recovery RL [8] to avoid constraint violations.

15.0.2 Hyperparameters for MESA and Comparisons

We report global hyperparameters shared across all algorithms in Table 15.1 and additionally include domain specific hyperparameters in separate tables in Tables 15.2, 15.3, and 15.4. We use the same base neural network architecture for the safety critic, recovery policy, actor for the task policy, and critic for the task policy. This base network is a fully connected network with 2 hidden layers each with 256 hidden units. For the task policy, we utilize the Soft Actor Critic algorithm from [62] and build on the implementation provided in [236].

15.0.3 Dataset Details

To collect datasets from the training environments, we train SAC on each of the training environments and log the replay buffer from an intermediate checkpoint. For the HalfCheetah-Disabled and Ant-Disabled tasks, we collect 4 and 3 training datasets of 400 episodes (on average ∼400K transitions with 14K and 113K violations) respectively. The dataset from the testing environment consists of 40K transitions (2.4K, and 11.2K violations for HalfCheetah, Ant), which is **10x** smaller than before. For the Cartpole-Length task, 20 training datasets are generated, with each containing 200 episodes of data (∼20K timesteps with 4.5K violations). The dataset from the testing environment contains 1K transitions (with 200 violations), which is **20x** smaller than before.

Algorithm 5 MEta-learning for Safe Adaptation (MESA)

```
Require: Training datasets $\mathcal{D}^{\text{train}} = \{\mathcal{D}_i^{\text{train}}\}_{i=1}^{N_{\text{train}}}$, adaptation dataset $\mathcal{D}^{\text{test}}$, task horizon
  $H$, safety threshold $\epsilon_{\text{risk}}$, safety critic step sizes $\alpha_1$ and $\alpha_2$, recovery policy step size $\beta$.
  for $i \in \{1, \dots N\}$ do                                   ▷ Phase 1: Offline Meta-Learning
      for $j \in \{1, \dots K\}$ do
          Sample $\mathcal{D}_j^{\text{train}} \sim \mathcal{D}^{\text{train}}$
          $\psi'_j \leftarrow \psi - \alpha_1 \cdot \nabla_\psi \mathcal{L}_{\text{risk}}\left(\psi, \mathcal{D}_j^{\text{train}}\right)$
      end for
      for $j \in \{1, \dots K\}$ do
          Sample $\mathcal{D}_j^{\text{test}} \sim \mathcal{D}^{\text{test}}$
      end for
      $\psi \leftarrow \psi - \alpha_2 \cdot \sum_j \nabla_\psi \mathcal{L}_{\text{risk}}\left(\psi'_j, \mathcal{D}_j^{\text{train}}\right)$
      $\omega \leftarrow \omega - \beta \cdot \nabla_\omega \mathcal{L}_{\pi_{\text{rec}}}\left(\omega, \mathcal{D}^{\text{test}}\right)$
  end for
  for $i \in \{1, \dots M\}$ do                                   ▷ Phase 2: Test Time Adaptation
      $\psi \leftarrow \psi - \alpha_1 \cdot \nabla_\psi \mathcal{L}_{\text{risk}}\left(\psi, \mathcal{D}^{\text{test}}\right)$
      $\omega \leftarrow \omega - \beta \cdot \nabla_\omega \mathcal{L}_{\pi_{\text{rec}}}\left(\omega, \mathcal{D}^{\text{test}}\right)$
  end for
  $\mathcal{D}^{\text{task}} \leftarrow \emptyset$
  while not converged do                                          ▷ Phase 3: Recovery RL
      $s_1 \sim$ env.reset()
      for $t \in \{1, \dots H\}$ do
          $a_t^\pi, a_t^{\text{rec}} \sim \pi_\theta(\cdot|s_t), \pi_{\text{rec}}(\cdot|s_t)$
          if $Q_{\text{risk}}^\pi(s_t, a_t) \le \epsilon_{\text{risk}}$ then
              $a_t = a_t^\pi$
          else
              $a_t = a_t^{\text{rec}}$
          end if
          Execute $a_t$, observe $r_t$, $c_t$, and $s_{t+1}$
          Add $(s_t, a_t^\pi, c_t, s_{t+1})$ to $\mathcal{D}^{\text{test}}$
          Add $(s_t, a_t, r_t, s_{t+1})$ to $\mathcal{D}^{\text{task}}$
          $\theta \leftarrow \theta - \gamma \cdot \nabla_\theta \mathcal{L}_\pi\left(\theta, \mathcal{D}^{\text{task}}\right)$
          $\psi \leftarrow \psi - \alpha_1 \cdot \nabla_\psi \mathcal{L}_{\text{safe}}\left(\psi, \mathcal{D}^{\text{test}}\right)$
          $\omega \leftarrow \omega - \beta \cdot \nabla_\omega \mathcal{L}_{\pi_{\text{rec}}}\left(\omega, \mathcal{D}^{\text{test}}\right)$
          if $c_t$ then
              End episode
          end if
      end for
  end while
```

Hyperparameters	Unconstrained	RRL	Multi-Task	MESA
Phase 1: Offline Training ($\mathcal{D}_{\text{train}}$)				
Total Iterations	—	—	10000	10000
Inner Batch Size $\lvert B_{\text{in}}\rvert$	—	—	—	256
Outer Batch Size $\lvert B_{\text{out}}\rvert$	—	—	—	256
Inner Adaptation Steps	—	—	—	1
Inner LR α_1	—	—	—	0.001
Outer LR α_2	—	—	—	0.00001
Task Batch Size K	—	—	—	5
Adam LR η	—	—	0.0003	—
Batch Size B	—	—	256	—
Phase 2: Offline Finetuning ($\mathcal{D}_{\text{test}}$)				
Total Iterations M	—	10000	500	500
Batch Size B	—	256	256	256
Adam LR	—	0.0003	η	α_1
Phase 3: Online Finetuning				
Adam LR	0.0003	0.0003	η	α_1
Batch Size B	256	256	256	256
Discount γ	0.99	0.99	0.99	0.99
γ_{risk}	—	0.8	0.8	0.8
ϵ_{risk}	—	0.1	0.1	0.1

Table 15.1: Algorithm Hyperparameters.

Hyperparameters	Unconstrained	RRL	Multi-Task	MESA
Phase 2: Offline Finetuning ($\mathcal{D}_{\text{test}}$)				
Total Iterations M	—	2000	100	100
Batch Size B	—	64	64	64
Phase 3: Online Finetuning				
γ_{risk} (Navigation 2)	—	0.65	0.65	0.65
ϵ_{risk} (Navigation 1)	—	0.3	0.3	0.3

Table 15.2: Navigation Hyperparameter Differences

Hyperparameters	Unconstrained	RRL	Multi-Task	MESA
Phase 1: Offline Training ($\mathcal{D}_{\text{train}}$)				
Total Iterations	—	—	15000	15000

Table 15.3: HalfCheetah-Disabled Hyperparameter Differences.

Hyperparameters	Unconstrained	RRL	Multi-Task	MESA
Phase 1: Offline Training ($\mathcal{D}_{\text{train}}$)				
Total Iterations	—	—	15000	15000
Phase 3: Online Finetuning				
Risk Threshold ϵ_{risk}	—	0.3	0.3	0.3

Table 15.4: Ant-Disabled Hyperparameter Differences.

Dataset	N_{train}	$\|D_{\text{train}}\|$	$\|D_{\text{test}}\|$
Cartpole-Length	20	20K	1K
HalfCheetah-Disabled	4	400K	40K
Ant-Disabled	3	400K	40K

Table 15.5: Dataset Hyperparameters.

For all environments, datasets are collected by an early-stopped SAC run, where the episode does not end on constraint violation. The testing dataset is collected by a randomly initialized policy. Each episode consists of 1000 timesteps.

Chapter 16

Appendix for Chapter 10

Here we provide further details on our MuJoCo experiments, hyperparameter sensitivity, simulated fabric experiments, and physical fabric experiments.

16.0.1 MuJoCo

As stated in the main text, we evaluate on the HalfCheetah-v2, Walker2D-v2, and Ant-v2 environments. To train the algorithmic supervisor, we utilize the TD3 implementation from OpenAI SpinningUp (`https://spinningup.openai.com/en/latest/`) with default hyperparameters and run for 100, 200, and 500 epochs respectively. The expert policies obtain rewards of 5330.78 ± 117.65, 3492.08 ± 1110.31, and 4492.88 ± 1580.42, respectively. Note that the experts for Walker2D and Ant have high variance, resulting in higher variance for the corresponding learning curves in Figure 8.3. We provide the state space dimensionality $|S|$, action space dimensionality $|A|$, and LazyDAgger hyperparameters (see Algorithm 1) for each environment in Table 16.1. The β_H value in the table is multiplied with the maximum possible action discrepancy $||a_{\text{high}} - a_{\text{low}}||_2^2$ to become the threshold for training $f(\cdot)$. In MuJoCo environments, $a_{\text{high}} = \vec{1}$ and $a_{\text{low}} = -\vec{1}$. The β_H value used for SafeDAgger in all experiments is chosen by the method provided in the paper introducing SafeDAgger [19]: the threshold at which roughly 20% of the initial offline dataset is classified as "unsafe."

For LazyDAgger and all baselines, the actor policy $\pi_R(\cdot)$ is a neural network with 2 hidden layers with 256 neurons each, rectified linear unit (ReLU) activation, and hyperbolic tangent output activation. For LazyDAgger and SafeDAgger, the discrepancy classifier $f(\cdot)$ is a neural network with 2 hidden layers with 128 neurons each, ReLU activation, and sigmoid output activation. We take 2,000 gradient steps per epoch and optimize with Adam and learning rate 1e-3 for both neural networks. To collect $\mathcal{D}$ and $\mathcal{D}_S$ in Algorithm 1 and SafeDAgger, we randomly partition our dataset of 4,000 state-action pairs into 70% (2,800 state-action pairs) for $\mathcal{D}$ and 30% (1,200 state-action pairs) for $\mathcal{D}_S$.

Environment	$\lvert S\rvert$	$\lvert A\rvert$	N	T	β_H	β_R	σ^2
HalfCheetah	16	7	10	5000	5e-3	β_H / 10	0.30
Walker2D	16	7	15	5000	5e-3	β_H / 10	0.10
Ant	111	8	15	5000	5e-3	β_H / 2	0.05

Table 16.1: **MuJoCo Hyperparameters:** $|S|$ and $|A|$ are aspects of the Gym environments while the other values are hyperparameters of LazyDAgger (Algorithm 1). Note that T and β_H are the same across all environments, and that β_R is a function of β_H.

16.0.2 LazyDAgger Switching Thresholds

As described in Section 8.5.1, the main LazyDAgger hyperparameters are the safety thresholds for switching to supervisor control (β_H) and returning to autonomous control (β_R). To tune these hyperparameters in practice, we initialize β_H and β_R with the method in Zhang et al. [19]; again, this sets the safety threshold such that approximately 20% of the initial dataset is unsafe. We then tune β_H higher to balance reducing the frequency of switching to the supervisor with allowing enough supervision for high policy performance. Finally we set β_R as a multiple of β_H, starting from $\beta_R = \beta_H$ and tuning downward to balance improving the performance and increasing intervention length with keeping the total number of actions moderate. Note that since these parameters are not automatically set, we must re-run experiments for each change of parameter values. Since this tuning results in unnecessary supervisor burden, eliminating or mitigating this requirement is an important direction for future work.

To analyze sensitivity to β_R and β_H, we plot the results of a grid search over parameter values on each of the MuJoCo environments in Figure 16.1. Note that a lighter color in the heatmap is more desirable for reward while a darker color is more desirable for actions and switches. We see that the supervisor burden in terms of actions and context switches is not very sensitive to the threshold as we increase β_H but jumps significantly for the very low setting ($\beta_H = 5 \times 10^{-4}$) as a large amount of data points are classified as unsafe. Similarly, we see that reward is relatively stable (note the small heatmap range for HalfCheetah) as we decrease β_H but can suffer for high values, as interventions are not requested frequently enough. Reward and supervisor burden are not as sensitive to β_R but follow the same trends we expect, with higher reward and burden as β_R decreases.

16.0.3 Fabric Smoothing in Simulation

Fabric Simulator

More information about the fabric simulator can be found in Seita et al. [228], but we review the salient details here. The fabric is modeled as a mass-spring system with a $n \times n$ square grid of point masses. Self-collision is implemented by applying a repulsive force between points

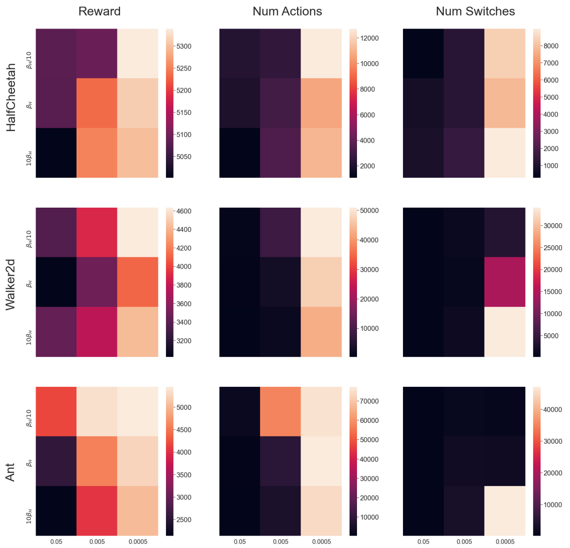


Figure 16.1: LazyDAgger β_R and β_H sensitivity heatmaps across the 3 MuJoCo environments. The x-axis denotes β_H and the y-axis denotes β_R. Note that β_R is a function of β_H. Each of the 3 environments was run 9 times with the different settings of β_R and β_H. As in Figure 8.3 we plot test reward, number of supervisor actions, and number of context switches.

that are sufficiently close together. Blender (`https://blender.org/`) is used to render the fabric in $100 \times 100 \times 3$ RGB image observations. See Figure 8.4 for an example observation. The actions are 4D vectors consisting of a pick point $(x, y) \in [-1, 1]^2$ and a place point $(\Delta x, \Delta y) \in [-1, 1]^2$, where $(x, y) = (-1, -1)$ corresponds to the bottom left corner of the plane while $(\Delta x, \Delta y)$ is multiplied by 2 to allow crossing the entire plane. In simulation, we initialize the fabric with coverage $41.1 \pm 3.4\%$ in the hardest (Tier 3) state distribution in [228] and end episodes if we exceed 10 time steps, cover at least 92% of the plane, are at least 20% out of bounds, or have exceeded a tearing threshold in one of the springs. We use the same algorithmic supervisor as [228], which repeatedly picks the coordinates of the corner furthest from its goal position and pulls toward this goal position. To facilitate transfer to the real world, we use the domain randomization techniques in [228] to vary the following parameters:

- Fabric RGB values uniformly between (0, 0, 128) and (115, 179, 255), centered around blue.
- Background plane RGB values uniformly between (102, 102, 102) and (153, 153, 153).
- RGB gamma correction uniformly between 0.7 and 1.3.
- Camera position (x, y, z) as $(0.5+\delta_1, 0.5+\delta_2, 1.45+\delta_3)$ meters, where each δ_i is sampled from $\mathcal{N}(0, 0.04)$.
- Camera rotation with Euler angles sampled from $\mathcal{N}(0, 90°)$.

- Random noise at each pixel uniformly between -15 and 15.

For consistency, we use the same domain randomization in our sim-to-sim ("simulator to simulator") fabric smoothing experiments in Section 8.6.2.

Actor Policy and Discrepancy Classifier

The actor policy is a convolutional neural network with the same architecture as [228], i.e. four convolutional layers with 32 3x3 filters followed by four fully connected layers. The parameters, ignoring biases for simplicity, are:

```
policy/convnet/c1   864 params (3, 3, 3, 32)
policy/convnet/c2   9216 params (3, 3, 32, 32)
policy/convnet/c3   9216 params (3, 3, 32, 32)
policy/convnet/c4   9216 params (3, 3, 32, 32)
policy/fcnet/fc1    3276800 params (12800, 256)
policy/fcnet/fc2    65536 params (256, 256)
policy/fcnet/fc3    65536 params (256, 256)
policy/fcnet/fc4    1024 params (256, 4)
Total model parameters: 3.44 million
```

The discrepancy classifier reuses the actor's convolutional layers by taking a forward pass through them. We do not backpropagate gradients through these layers when training the classifier, but rather fix these parameters after training the actor policy. The rest of the classifier network has three fully connected layers with the following parameters:

```
policy/fcnet/fc1    3276800 params (12800, 256)
policy/fcnet/fc2    65536 params (256, 256)
policy/fcnet/fc3    1024 params (256, 4)
Total model parameters: 3.34 million
```

Training

Due to the large amount of data required to train fabric smoothing policies, we pretrain the *actor policy* (not the discrepancy classifier) in simulation. The learned policy is then fine-tuned to the new environment while the discrepancy classifier is trained from scratch. Since the algorithmic supervisor can be queried cheaply, we pretrain with DAgger as in [228]. To further accelerate training, we parallelize environment interaction across 20 CPUs, and before DAgger iterations we pretrain with 100 epochs of Behavior Cloning on the dataset of 20,232 state-action pairs available at [228]'s project website. Additional training hyperparameters are given in Table 16.2 and the learning curve is given in Figure 16.2.

Experiments

In sim-to-sim experiments, the initial policy is trained on a 16x16 grid of fabric in a range of colors centered around blue with a spring constant of $k = 10,000$. We then adapt this

Hyperparameter	Value
BC Epochs	100
DAgger Epochs	100
Parallel Environments	20
Gradient Steps per Epoch	240
Env Steps per Env per DAgger Epoch	20
Batch Size	128
Replay Buffer Size	5e4
Learning Rate	1e-4
L2 Regularization	1e-5

Table 16.2: **DAgger Hyperparameters.** After Behavior Cloning, each epoch of DAgger (1) runs the current policy and collects expert labels for 20 time steps in each of 20 parallel environments and then (2) takes 240 gradient steps on minibatches of size 128 sampled from the replay buffer.

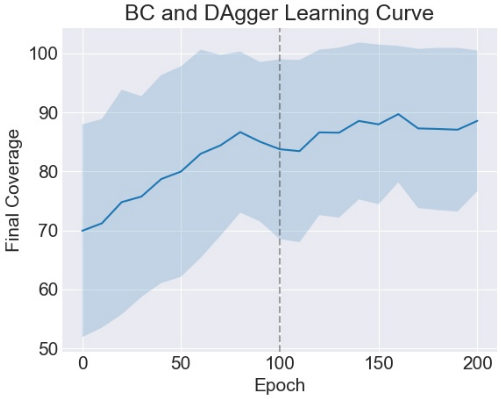


Figure 16.2: Behavior Cloning and DAgger performance across 10 test episodes evaluated every 10 epochs. Shading indicates 1 standard deviation. The first 100 epochs (left half) are Behavior Cloning epochs and the second 100 (right half) are DAgger epochs.

policy to a new simulator with different physics parameters and an altered visual appearance. Specifically, in the new simulation environment, the fabric is a higher fidelity 25x25 grid with a lower spring constant of $k = 2,000$ and a color of (R, G, B) = (204, 51, 204) (i.e. pink), which is outside the range of colors produced by domain randomization (Section 16.0.3). Hyperparameters are given in Table 16.3.

16.0.4 Fabric Manipulation with the ABB YuMi

Experimental Setup

We manipulate a brown 10” by 10” square piece of fabric with a single parallel jaw gripper as shown in Figure 8.1. The gripper is equipped with reverse tweezers for more precise picking of

Hyperparameter	Value
N	10
T	20
β_H	0.001
β_R	β_H
σ^2	0.05
Initial $\lvert\mathcal{D}\rvert$	1050
Initial $\lvert\mathcal{D}_S\rvert$	450
Batch Size	50
Gradient Steps per Epoch	200
π Learning Rate	1e-4
f Learning Rate	1e-3
L2 Regularization	1e-5

Table 16.3: Hyperparameters for sim-to-sim fabric smoothing experiments, where the first 5 rows are LazyDAgger hyperparameters in Algorithm 1. Initial dataset sizes and batch size are in terms of images *after* data augmentation, i.e. scaled up by a factor of 15 (see Section 16.0.4). Note that the offline data is split 70%/30% as in Section 16.0.1.

deformable materials. Neural network architecture is consistent with Section 16.0.3 for both actor and safety classifier. We correct pick points that nearly miss the fabric by mapping to the nearest point on the mask of the fabric, which we segment from the images by color. To convert neural network actions to robot grasps, we run a standard camera calibration procedure and perform top-down grasps at a fixed depth. By controlling the width of the tweezers via the applied force on the gripper, we can reliably pick only the top layer of the fabric at a given pick point. We provide LazyDAgger-Execution hyperparameters in Table 16.4.

Image Processing Pipeline

In the simulator, the fabric is smoothed against a light background plane with the same size as the fully smoothed fabric (see Figure 8.4). Since the physical workspace is far larger than the fabric, we process each RGB image of the workspace by (1) taking a square crop, (2) rescaling to 100 × 100, and (3) denoising the image. Essentially we define a square crop of the workspace as the region to smooth and align against, and assume that the fabric starts in this region. These processed images are the observations that fill the replay buffer and are passed to the neural networks.

User Interface

When the system solicits human intervention, an interactive user interface displays a scaled-up version of the current observation. The human is able to click and drag on the image to

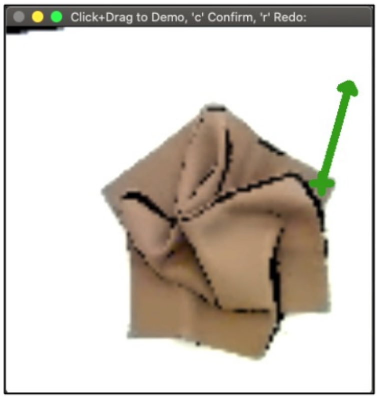


Figure 16.3: The user interface for human interventions. The current observation of the fabric state from the robot's perspective is displayed, with an overlaid green arrow indicating the action the human has just specified.

provide a pick point and pull vector, respectively. The interface captures the input as pixel locations and analytically converts it to the action space of the environment (i.e. $a \in [-1, 1]^4$) for the robot to execute. See Figure 16.3 for a screen capture of the user interface.

Data Augmentation

To prevent overfitting to the small amount of real data, before adding each state-action pair to the replay buffer, we make 10 copies of it with the following data augmentation procedure, with transformations applied in a random order:

- Change contrast to 85-115% of the original value.
- Change brightness to 90-110% of the original value.
- Change saturation to 95-105% of the original value.
- Add values uniformly between -10 and 10 to each channel of each pixel.
- Apply a Gaussian blur with σ between 0 and 0.6.
- Add Gaussian noise with σ between 0 and 3.
- With probability 0.8, apply an affine transform that (1) scales each axis independently to 98-102% of its original size, (2) translates each axis independently by a value between -2% and 2%, and (3) rotates by a value between -5 and 5 degrees.

Hyperparameter	Value
β_H	0.004
β_R	β_H
$\lvert\mathcal{D}\rvert$	875
$\lvert\mathcal{D}_S\rvert$	375
Batch Size	50
Gradient Steps per Epoch	125
π Learning Rate	1e-4
f Learning Rate	1e-3
L2 Regularization	1e-5

Table 16.4: Hyperparameters for physical fabric experiments provided in the same format as Table 16.3. Since this is at execution time, N, T and σ^2 hyperparameters do not apply.