Scalable Supervision for Safe and Efficient Online Robot Learning

by

Ashwin Balakrishna

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

Spring 2022

Scalable Supervision for Safe and Efficient Online Robot Learning

Abstract

Scalable Supervision for Safe and Efficient Online Robot Learning

by

Ashwin Balakrishna

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Ken Goldberg, Chair

Robotics has seen increasing success in automating a wide variety of tasks in structured settings, such as factories and assembly lines. However, using robots for reliable automation in less structured, open-world environments remains a critical challenge. Recent advances in machine learning have played a critical role in the development of agents which can bridge this gap by exploring the environment online and leveraging collected experiences to modify their behavior. For these algorithms to be broadly applicable in practice, they must be both efficient enough to direct exploration towards regions that are relevant for learning new tasks while being safe and reliable enough to deploy in the physical world. However, this is exceedingly challenging if these algorithms must discover information about promising and unpromising behaviors entirely from their own online experience, and are not provided with any additional supervision indicating promising behaviors to emulate and unpromising behaviors to avoid.

This dissertation introduces a number of online learning algorithms for learning from demonstrations, reinforcement learning, and bandit exploration which use scalable sources of supervision, both from sparing human queries and offline datasets, to structure online exploration to facilitate safe and efficient robot learning. I will discuss both theoretical properties of these algorithms and their evaluation in a number of environments with uncertain dynamics both in simulation and on robotic manipulation tasks in the physical world. I will end by talking about opportunities for future work in improving the scalability and applicability of online learning algorithms for robot learning in practice.

To my Mom and Dad for their constant support through the many highs and lows of my time as a PhD student as Berkeley and for giving me the opportunities that gave me the freedom to pursue my curiosity throughout my education and life.

Contents

Contents **ii**

1 Introduction **1**
- 1.1 Efficient Online Imitation Learning . . . 2
- 1.2 Reinforcement Learning from Suboptimal Demonstrations . . . 3
- 1.3 Reinforcement Learning from Negative Demonstrations . . . 3
- 1.4 Learning Priors for Rapid Bandit-Based Grasp Exploration . . . 4

I Efficient Online Imitation Learning **6**

2 LazyDAgger: Reducing Context Switching in Interactive Imitation Learning **7**
- 2.1 Background and Related Work . . . 8
- 2.2 Problem Statement . . . 10
- 2.3 Preliminaries: SafeDAgger . . . 11
- 2.4 LazyDAgger . . . 12
  - 2.4.1 Action Discrepancy Prediction . . . 12
  - 2.4.2 Noise Injection . . . 13
- 2.5 Experiments . . . 14
  - 2.5.1 Simulation Experiments: MuJoCo Benchmarks . . . 14
  - 2.5.2 Fabric Smoothing in Simulation . . . 17
  - 2.5.3 Physical Fabric Manipulation Experiments . . . 18
- 2.6 Discussion and Future Work . . . 19

3 ThriftyDAgger: Budget-Aware Novelty and Risk Gating for Interactive Imitation Learning **21**
- 3.1 Related Work . . . 23
- 3.2 Problem Statement . . . 24
- 3.3 ThriftyDAgger . . . 25
  - 3.3.1 Novelty Estimation . . . 25
  - 3.3.2 Risk Estimation . . . 26

3.3.3 Regulating Switches in Control Modes . . . 26
3.3.4 Computing Risk and Novelty Thresholds from Data . . . 27
3.3.5 ThriftyDAgger Overview . . . 27
3.4 Experiments . . . 28
3.4.1 Evaluation Metrics . . . 28
3.4.2 Comparisons . . . 29
3.4.3 Peg Insertion in Simulation . . . 29
3.4.4 User Study: Controlling A Fleet of Three Robots in Simulation . . . 30
3.4.5 Physical Experiment: Visuomotor Cable Routing . . . 32
3.5 Discussion and Future Work . . . 32

4 On-Policy Robot Imitation Learning from a Converging Supervisor 34
4.1 Related Work . . . 35
4.2 Converging Supervisor Framework and Preliminaries . . . 36
4.2.1 On-Policy Imitation Learning . . . 36
4.2.2 Converging Supervisor Framework (CSF) . . . 37
4.3 Regret Analysis . . . 38
4.3.1 Static Regret . . . 38
4.3.2 Dynamic Regret . . . 39
4.4 Converging Supervisors for Deep Continuous Control . . . 41
4.5 Experiments . . . 42
4.5.1 Simulation Experiments . . . 43
4.5.2 Physical Robot Experiments . . . 43
4.6 Discussion and Future Work . . . 44

II Reinforcement Learning from Suboptimal Demonstrations 46

5 ABC-LMPC: Safe Learning MPC for Stochastic Nonlinear Dynamical Systems with Adjustable Boundary Conditions 47
5.1 Related Work . . . 48
5.2 Problem Statement . . . 49
5.3 Preliminaries . . . 50
5.3.1 Safe Set . . . 50
5.3.2 Value Function . . . 51
5.3.3 Transfer to Novel Goal Sets . . . 52
5.4 Controller Design . . . 52
5.4.1 Task Driven Optimization . . . 53
5.4.2 Start State Expansion . . . 54
5.5 Properties of ABC-LMPC . . . 55
5.6 Practical Implementation . . . 56
5.6.1 Sample-Based Safe Set . . . 56

5.6.2 Start State Expansion Strategy 57
5.6.3 Goal Set Transfer 57
5.6.4 ABC-LMPC Optimization Procedure 57
5.7 Experiments 58
5.7.1 Experimental Domains 58
5.7.2 Fixed Start and Goal Conditions 59
5.7.3 Start State Expansion 59
5.7.4 Goal Set Transfer 60
5.7.5 Inverted Pendulum Swing-Up Task 62
5.8 Discussion and Future Work 62

6 SAVED: Safe Deep Model-Based RL for Sparse Cost Robotic Tasks 63
6.1 Related work 65
6.2 Safety Augmented Value Estimation from Demonstrations (SAVED) 65
6.2.1 Assumptions and Preliminaries 66
6.2.2 Algorithm Overview 67
6.2.3 Task Completion Driven Exploration 68
6.2.4 Chance Constraint Enforcement 70
6.2.5 Algorithm Pseudocode 70
6.3 Experiments 70
6.3.1 Baselines 71
6.3.2 Simulated Navigation 72
6.3.3 Simulated Robot Experiments 74
6.3.4 Physical Robot Experiments 75
6.4 Discussion and Future Work 76

7 LS^3: Latent Space Safe Sets for Long-Horizon Visuomotor Control of Sparse Reward Iterative Tasks 77
7.1 Related Work 78
7.1.1 Safe, Iterative Learning Control 78
7.1.2 Model Based Reinforcement Learning 79
7.1.3 Reinforcement Learning from Pixels 80
7.2 Problem Statement 80
7.3 Latent Space Safe Sets (LS^3) 81
7.3.1 Learning a Latent Space for Planning 83
7.3.2 Latent Safe Sets for Model-Based Control 83
7.3.3 Reward and Constraint Estimation 84
7.3.4 Model-Based Planning with LS^3 84
7.4 Experiments 85
7.4.1 Comparisons 86
7.4.2 Evaluation Metrics 86
7.4.3 Domains 87

7.4.4 Simulation Results 87
7.4.5 Physical Results 89
7.5 Discussion and Future Work 89

8 Monte Carlo Augmented Actor-Critic for Sparse Reward Deep RL from Suboptimal Demonstrations 90
8.1 Related Work 91
8.1.1 Reinforcement Learning from Demonstrations 91
8.1.2 Improving Q-Value Estimates 92
8.2 Problem Statement 92
8.3 Preliminaries: Actor-Critic Algorithms 93
8.4 Monte Carlo augmented Actor-Critic 94
8.4.1 MCAC Algorithm 94
8.4.2 MCAC Practical Implementation 95
8.5 Experiments 96
8.5.1 Experimental Procedure 97
8.5.2 Domains 97
8.5.3 Algorithm Comparisons 100
8.5.4 Results 101
8.6 Discussion and Future Work 102

III Reinforcement Learning from Negative Demonstrations 103

9 Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones 104
9.1 Related Work 106
9.2 Problem Statement 108
9.3 Recovery RL 109
9.3.1 Preliminaries: Training a Safety Critic 109
9.3.2 Defining a Recovery Set and Policy 109
9.3.3 Offline Pretraining 111
9.3.4 Practical Implementation 112
9.4 Experiments 112
9.5 Discussion and Future Work 118

10 MESA: Offline Meta-RL for Safe Adaptation and Fault Tolerance 119
10.1 Related Work 120
10.1.1 Safe Reinforcement Learning 120
10.1.2 Meta Reinforcement Learning 121
10.2 Preliminaries 122
10.2.1 Constrained Markov Decision Processes 122
10.2.2 Safety Critics for Safe RL 122

10.2.3 Recovery RL 123
10.2.4 Meta-learning 123
10.3 Problem Statement 124
10.4 MEta-learning for Safe Adaptation (MESA) 124
10.4.1 Phase 1, Meta-Learning Q^{π}_{risk} 125
10.4.2 Phase 2, Test Time Adaptation 125
10.4.3 Phase 3, Using Q^{π}_{risk} and π_{rec} for Safe RL 125
10.5 Experiments 125
10.5.1 Data Collection 129
10.5.2 Results 130
10.6 Ablations 130
10.6.1 Test Dataset Size 131
10.6.2 Test Environment Generalization 131
10.7 Discussion and Future Work 131

IV Learning Priors for Rapid Bandit-Based Grasp Exploration 133

11 Accelerating Grasp Exploration by Leveraging Learned Priors 134
11.1 Related Work 136
11.2 Problem Statement 137
11.2.1 Assumptions 137
11.2.2 Definitions 137
11.2.3 Bayesian Bandits 138
11.2.4 Learning Objective 138
11.3 Grasp Exploration Method 138
11.3.1 Thompson Sampling with a Beta-Bernoulli Process 139
11.3.2 Leveraging Neural Network Priors 139
11.3.3 Prior Mismatch 140
11.4 Practical Implementation 141
11.5 Experiments 142
11.5.1 Setup 142
11.5.2 Simulation Experiments 144
11.6 Discussion and Conclusion 146
11.7 Discussion and Future Work 146

12 Exploratory Grasping: Asymptotically Optimal Algorithms for Grasping Challenging Polyhedral Objects 148
12.1 Related Work 149
12.2 Exploratory Grasping: Problem Statement 150
12.2.1 Exploratory Grasping as an MDP 151
12.2.2 Assumptions 152

12.2.3 Learning Objective 152
12.3 Reinforcement Learning for Exploratory Grasping 153
12.3.1 Analyzing the Exploratory Grasping MDP 153
12.4 Bandits for Online Rapid Grasp Exploration Strategy (BORGES) 154
12.5 Experiments 157
12.5.1 Simulation Experiments 157
12.5.2 Initial Physical Experiments 159
12.6 Future Work 160

V Conclusion and Future Work 161

13 Conclusion 162
13.1 Opportunities for Future Work 163
13.1.1 Efficient Online Imitation Learning 163
13.1.2 Reinforcement Learning from Suboptimal Demonstrations 163
13.1.3 Reinforcement Learning from Negative Demonstrations 164
13.1.4 Learning Priors for Rapid Bandit-Based Grasp Exploration 164
13.2 Broader Vision for Robot Learning 164

Bibliography 166

A LazyDAgger: Reducing Context Switching in Interactive Imitation Learning 190
A.1 MuJoCo 190
A.2 LazyDAgger Switching Thresholds 191
A.3 Fabric Smoothing in Simulation 192
A.4 Fabric Manipulation with the ABB YuMi 195

B ThriftyDAgger: Budget-Aware Novelty and Risk Gating for Interactive Imitation Learning 198
B.1 Algorithm Details 198
B.1.1 ThriftyDAgger 198
B.1.2 Behavior Cloning 199
B.1.3 SafeDAgger 199
B.1.4 LazyDAgger 199
B.1.5 HG-DAgger 201
B.2 Hyperparameter and Implementation Details 201
B.2.1 ThriftyDAgger 201
B.2.2 Behavior Cloning 202
B.2.3 SafeDAgger 202
B.2.4 LazyDAgger 203

B.2.5 HG-DAgger . . . 203
B.3 Environment Details and Additional Metrics . . . 203
B.3.1 Peg Insertion in Simulation . . . 203
B.3.2 Block Stacking in Simulation . . . 204
B.3.3 Physical Cable Routing . . . 205
B.4 User Study Details . . . 207
B.4.1 User Study Interface . . . 207
B.4.2 NASA TLX Survey Results . . . 208
B.4.3 Wall Clock Time . . . 208
B.4.4 Detailed Protocol . . . 208

C On-Policy Robot Imitation Learning from a Converging Supervisor 210
C.1 Static Regret . . . 210
C.1.1 Proof of Theorem 4.3.1 . . . 210
C.1.2 Proof of Corollary 4.3.1 . . . 212
C.2 Dynamic Regret . . . 213
C.2.1 Proof of Lemma 4.3.1 . . . 213
C.2.2 Proof of Corollary 4.3.2 . . . 215
C.2.3 Predictability of Online Learning Problems . . . 215
C.2.4 Proof of Lemma 4.3.2 . . . 218
C.2.5 Proof of Theorem 4.3.2 . . . 219
C.3 Training Details . . . 219
C.3.1 CSF Learner . . . 219
C.3.2 PETS . . . 219
C.3.3 SAC . . . 220
C.3.4 TD3 . . . 220
C.3.5 ME-TRPO . . . 220
C.4 Experimental Details . . . 220
C.4.1 Simulated Experiments . . . 220
C.4.2 Physical Experiments . . . 221

D Safe Learning MPC for Stochastic Nonlinear Dynamical Systems with Adjustable Boundary Conditions 222
D.1 Proofs of Controller Properties . . . 222
D.2 Adjustable Boundary Condition LMPC Implementation Details . . . 225
D.2.1 Solving the MPC Problem . . . 225
D.2.2 Value Function . . . 225
D.2.3 Start State Expansion . . . 225
D.3 Experiment Specific Parameters . . . 226
D.3.1 Pointmass Navigation . . . 226
D.3.2 7-Link Reacher Arm . . . 226
D.3.3 Inverted Pendulum . . . 227

D.4 Controller Domain Expansion Strategy 227

E SAVED: Safe Deep Model-Based RL for Sparse Cost Robotic Tasks 229
E.1 Additional Experimental Details for SAVED and Baselines 229
E.1.1 SAVED 229
E.1.2 Behavior Cloning 230
E.1.3 PETSfD and PETSfD Dense 230
E.1.4 SACfD 230
E.1.5 OEFD 231
E.2 Simulated Experiments Additional Results 231
E.3 Physical Experiments: Additional Details and Experiments 232
E.3.1 Figure-8 233
E.3.2 Knot-Tying 235
E.4 Ablations 236
E.4.1 SAVED 236
E.4.2 Model-Free 239

F LS^3: Latent Space Safe Sets for Long-Horizon Visuomotor Control of Sparse Reward Iterative Tasks 241
F.1 Algorithm Details 241
F.1.1 Latent Space Safe Sets (LS^3) 241
F.1.2 Soft Actor-Critic from Demonstrations (SACfD) 243
F.1.3 Soft Actor-Critic from Demonstrations with Learned Recovery Zones (SACfD+RRL) 244
F.1.4 Advantage Weighted Actor-Critic (AWAC) 244
F.2 LS^3 Implementation Details 244
F.3 Experimental Domain Details 245
F.3.1 Navigation 245
F.3.2 Reacher 246
F.3.3 Sequential Pushing 246
F.3.4 Physical Cable Routing 246
F.4 Additional Results 247
F.5 Sensitivity Experiments 247

G Monte Carlo Augmented Actor-Critic for Sparse Reward Deep RL from Suboptimal Demonstrations 249
G.1 Implementation Details 249
G.1.1 Behavioral Cloning 249
G.1.2 Twin Delayed Deep Deterministic Policy Gradients 249
G.1.3 Soft Actor Critic 249
G.1.4 Overcoming Exploration with Demonstrations 250
G.1.5 Advantage Weighted Actor Critic 250

H Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones 251
H.1 Recovery RL Theoretical Motivation and Variants . . . 251
H.1.1 Theoretical Motivation . . . 251
H.1.2 Safety Value Function . . . 252
H.1.3 Reachability-based Variant . . . 252
H.2 Algorithm Details . . . 252
H.2.1 Recovery RL . . . 252
H.2.2 Unconstrained . . . 253
H.2.3 Lagrangian Relaxation (LR) . . . 253
H.2.4 Risk Sensitive Policy Optimization (RSPO) . . . 253
H.2.5 Safety Q-Functions for Reinforcement Learning (SQRL) . . . 254
H.2.6 Reward Penalty (RP) . . . 254
H.2.7 Off Policy Reward Constrained Policy Optimization (RCPO) . . . 254
H.3 Additional Experimental Metrics . . . 254
H.4 Safety Critic Visualizations . . . 256
H.5 Implementation Details . . . 256
H.5.1 Network Architectures . . . 257
H.5.2 Global Training Details . . . 259
H.5.3 Recovery Policy Training Details . . . 260
H.6 Environment Specific Algorithm Parameters . . . 261
H.7 Environment Details . . . 261
H.7.1 Navigation Environments . . . 262
H.7.2 Manipulation Environments . . . 262
H.7.3 Image Maze . . . 263
H.7.4 Physical Experiments . . . 263
H.7.5 Additional Physical Experiment . . . 265

I MESA: Offline Meta-RL for Safe Adaptation and Fault Tolerance 266
I.1 Algorithm Description . . . 266
I.2 Hyperparameters for MESA and Comparisons . . . 266
I.3 Dataset Details . . . 266

J Exploratory Grasping: Asymptotically Optimal Algorithms for Grasping Challenging Polyhedral Objects 270
J.1 Proofs . . . 270
J.1.1 Proof of Lemma 12.3.1 . . . 270
J.1.2 Proof of Theorem 12.3.1 . . . 271
J.1.3 Proof of Theorem 12.4.1 . . . 272
J.1.4 Proof of Theorem 12.4.2 . . . 273
J.2 Experimental Details . . . 274
J.3 Additional Simulation Experiments . . . 274
J.4 Sensitivity Experiments . . . 275

J.4.1 Sensitivity to Number of Grasp Samples 275
J.4.2 Sensitivity to Exploratory Grasping Parameters 276
J.5 Initial Physical Experiments Details 277

Acknowledgments

My time at Berkeley has felt like a whirlwind of experiences which has helped me develop significantly as a researcher and a person in general. However, nothing I've done at Berkeley would have been possible without the support of a number of key mentors early on in life who helped prepare me for a career in research. To my high school physics teacher Mr. Chuck Williams, thank you for teaching me how to break down complex problems and for all your philosophical life advice. Thank you for also encouraging me to help deepen my understanding of ideas by explaining them to others. This has been a technique that has consistently helped me throughout my academic work, beginning with as a teaching assistant for your course, but also as a student and a researcher when trying to ensure my knowledge was thorough and robust. To Manuel Lopez, thank you for giving me the opportunity to work under you in the Aerospace Computing Lab when I was a clueless high school student. You gave me my first introduction to open-ended scientific exploration, and your encouragement, feedback, and advice gave me the knowledge and confidence to explore research as an undergraduate at Caltech.

As a freshman at Caltech, I was excited about getting involved in research, but still didn't have many of the skills needed to contribute meaningfully to most research projects. After applying for positions in a number of different labs, the only person who offered me a position was Professor Hyuck Choo, who seemed more interested in my enthusiasm for research than in my specific skillset. Thank you so much for taking a chance on me and giving me the freedom to explore a wide range of different projects. Your consistently thoughtful research feedback was always mixed with a jovial and relaxed attitude which made me feel consistently welcome, valued, and happy. I deeply appreciate your advice both on research and life, and you are the example I've always striven to emulate when interacting with my fellow researchers. I would also like to thank Jeong Oen Lee for working closely with me and mentoring me throughout my time in the Choo lab. You were my first role model in research, and I am grateful for all the time you spent teaching me both about technical concepts and how academic research works in general. Both you and Professor Choo made research both accessible and fun, and were my primary inspirations for embarking on a PhD.

When I first arrived at Berkeley for the EECS PhD visit days, I found the idea of robotics research exciting, but really didn't have much prior experience and was still trying to figure out what sorts of problems I was interested in. However, when I visited Professor Ken Goldberg's lab, his infectious enthusiasm for robotics and the friendly and welcoming students running robot demos got me excited about joining the AUTOLAB. This turned out to be one of the decisions that made the rest of my PhD both intellectually and personally rewarding, and I am incredibly grateful to Ken for giving me the opportunity to join the AUTOLAB despite my limited experience coming in. Ken's mentorship and advice has played a significant role in my development as a researcher, and I particularly appreciate him giving me the freedom to explore a variety of ideas while helping me learn how to shape these ideas into concrete research contributions through careful analysis of both problem assumptions and experimental data. Ken's extremely thorough advice on paper drafts and

conference talks/videos has also helped me develop my communication skills significantly. His insights have not only helped advance my research career, but will definitely be a valuable asset throughout my professional life.

When I first joined the AUTOLAB and was looking for inspiration on research to get started with, I had the good fortune to work with Mike Danielczuk and Matt Matl, who were my first collaborators in the AUTOLAB. Thank you for giving me such a great welcome to the lab – during my first few months at Berkeley, you both not only taught me a ton about robotics, but also made spending time in lab a super enjoyable experience. From the daily lunches on the 5th floor of Soda Hall to laughing at random YouTube videos when we should have been doing experiments, working with you was both extremely educational and highly entertaining. Matt ended up leaving the AUTOLAB soon after to play a major role in founding Ambi Robotics, but I'd like to additionally thank Mike for continuing to be such a great collaborator and friend throughout the rest of my PhD. I've always striven to learn as much as I could from your prowess with robot hardware, clarity of thought, and amazing ability to rapidly conduct experiments and carefully analyze their results. Working with you has helped me significantly in developing as a researcher, and your fun-loving spirit has always made working with you a blast.

After a few months in the AUTOLAB, my research interests started to shift towards reinforcement and imitation learning, and this led to a number of fun initial discussions with Brijen Thananjeyan. These discussions ended up leading to the most productive and enjoyable collaboration during my PhD, and my work with Brijen has formed the backbone of the work in this dissertation. Thank you for always being around for endless discussions and brainstorming on random research ideas, however wild and far-fetched, and for working closely with me on so many research projects. Developing as a researcher along with you has been so much fun, and our collaboration has been the highlight of my research career. I've always admired your research creativity, especially your ability to come up with ideas that are both elegant and practical, and have tried to learn from your calm, collected, and thorough approach to solving problems. I've also been very fortunate to get to know you well beyond our research discussions. From bingeing tennis matches and movies during late nights in the lab, to getting destroyed by you in tennis in real life, your friendship has significantly enriched my time at Berkeley.

I have also had the opportunity to work closely with a number of undergraduate researchers and masters students throughout my time at Berkeley. I appreciate your patience as I learned how best to help advise you, and would like to especially highlight Ryan Hoque, Albert Wilcox, Michael Luo, Leitan Fu, and Katherine Li for their significant contributions to the work in this dissertation. I was lucky to have the opportunity to continue to work with Ryan when he joined the AUTOLAB as a PhD student and quite extensively with Albert throughout his undergraduate work. To Ryan and Albert, I've always been impressed not only with your amazing abilities as researchers, but also with your relaxed and cheerful attitude towards research and life even during the most stressful of times. I've also been impressed with Albert's consistently low quality food recommendations and frequent comic relief. I would like to additionally thank Priya Sundaresan, Jennifer Grannen, Aditya Gana-

pathi, Vainavi Viswanath, Shivin Devgon, Zaynah Javed, and Satvik Sharma who I also worked closely with on a number of different projects focused on deformable manipulation, industrial automation, and reinforcement learning. While these collaborations are not directly discussed in this dissertation, I am grateful to have had the opportunity to advise, collaborate with, and learn from all of you. To Priya and Jennifer, you two were the first students I played a major role in advising; thank you for putting up with my initial incompetence as an advisor and for your friendship outside of the lab. I'm now always excited to hear about all of the awesome things you two are working on during your own PhDs.

There are also a number of others in the AUTOLAB who deserve recognition for making my research and personal experiences in the lab more enjoyable. To Jeffrey Ichnowski, Daniel Brown, Ajay Tanwani, Ellen Novoseller, Daniel Seita, and Minho Hwang, thank you for your advice, paper edits, and all the wisdom you've shared during our collaborations. I'd like to give a special shoutout to Jeff for his hilarious acronyms for algorithm names, the high quality Trader Joe's snacks he always brought to lab, and for teaching me so much about how to give research talks. To Justin Kerr and Alejandro Escontrela, your constant readiness with a joke and enthusiasm provided me with endless entertainment during the final year of my PhD, and I'm excited to hear about all of your work in the future. I would also like to thank all of the other members of the AUTOLAB I didn't have the opportunity to work directly with, especially Chung Min Kim, Raven Huang, and Yahav Avigal, for promoting such a friendly and light-hearted atmosphere. Struggling through research was significantly more enjoyable when surrounded by others with a broad perspective on life beyond the many ups and downs typical of life as a researcher.

While my friends and colleagues in the AUTOLAB were central to my PhD experience, there are a number of others at Berkeley and beyond who also played a significant role in my research journey. Early on in my PhD, I had a great time working with collaborators Andrey Kurenkov, Roberto Martín Martín, Animesh Garg, and Silvio Savarase at Stanford on identifying and extracting various objects from cluttered environments. This collaboration formed my first exploration into robotics research and taught me a lot about how to be an effective researcher in the AUTOLAB. I would also like to acknowledge my collaborators during my internship at Toyota Research Institute (TRI): Rowan McAllister, Blake Wulfe, Logan Ellis, Jean Mercat, and Adrien Gaidon. Thank you for making my time at TRI so enjoyable, giving me the opportunity to learn about so much exciting research, and providing both freedom and guidance as I explored a variety of different ideas in reinforcement learning and representation learning. Jonathan Lee, Felix Li, Arsh Zahed, Ugo Rosolia, Francesco Borrelli, Sergey Levine, Chelsea Finn, Anca Dragan, Claire Tomlin, Joey Gonzalez, and my collaborators at Google, Julian Ibarz and Jie Tan, also deserve significant acknowledgement for their insights and contributions on a number of projects on safe reinforcement learning and control. I am especially grateful to Ugo and Joey for their extensive advice and feedback on a number of different ideas early on in my PhD and to Claire, Anca, and Chelsea for serving on my qualifying exam and dissertation committees. In addition, I would like to thank all of the Berkeley EECS and BAIR administrators for their work behind the scenes in facilitating my experience here, especially Angie Abbatecola, Shirley Salanio, and Jean

Nguyen.

Finally, I must acknowledge my friends and family for their critical role in supporting me throughout my time at Berkeley. There are a large number of people who have played a significant role in making my life as a PhD student more enjoyable, but I would like to especially acknowledge Suraj Nair, Vansh Kumar, and Anshul Ramachandran for all of the fun times we have shared both before and during my time at Berkeley. I would like to additionally thank Suraj for being my go-to person to discuss both research ideas and life ideas and Anshul for referring me to my new job after Berkeley! Last, but certainly not least, I must thank my parents. Anything I have accomplished in my life is primarily due to your advice and support. Thank you for encouraging me to dream big and explore my interests while still helping me maintain perspective on the many other joys of life beyond academic pursuits.

Chapter 1

Introduction

While robots have already seen widespread adoption in structured manufacturing settings such as factories and assembly lines, their adoption in less structured settings such as households and city streets remains elusive. In order to bridge this gap, robots must have the ability to reason about large degrees of variability and uncertainty in their environment. One recent paradigm for addressing this challenge involves collecting or leveraging existing large offline datasets of interactive behaviors to learn robotic control policies which can be reliably deployed in the physical world. While these approaches have shown significant success in recent literature [1, 2, 3, 4, 5, 6, 7], I argue that for robots to be reliably and robustly deployed in uncertain environments, they must also have the ability to autonomously improve and adjust their control strategies through online exploration. This ability is critical for a number of reasons. First, there will often be significant domain shift between offline data sources and data collected in the environment in the robot is deployed, making online adaptation critical. Second, even in the absence of dynamics shift, the data distribution in offline datasets may not fully cover the space of behaviors relevant for the robot to learn how to do some new task it is given to learn. Finally, online learning makes it possible for robots to continually improve their performance as they execute their policies in the environment.

Developing algorithms to enable robots to effectively learn new tasks through online exploration in the real world remains an open challenge. Algorithmic paradigms from the online learning community such as imitation learning, reinforcement learning, and multi-armed bandits have emerged as popular approaches for learning tasks through online interaction. However, two core challenges which limit application of these ideas to learning robotic control policies is the difficulty in achieving (1) efficiency and (2) safety during online exploration. To apply online learning algorithms to robotics, algorithms must allow robots to learn performant policies with minimal environment interaction due to the significant time and monetary costs of operating robots in physical environments. Additionally, they must allow robots to explore their environments safely, and avoid engaging in behaviors which could damage themselves, their surroundings, or require excessive human intervention. Achieving these properties is exceedingly challenging when online learning algorithms are expected to identify promising and unpromising behaviors *entirely* from autonomous experience, and are provided no additional supervision indicating which behaviors may be good to emulate and which behaviors may be wiser to avoid.

This dissertation contributes a number of online learning algorithms which introduce methods to use scalable sources of supervision to structure online exploration to enable safe and efficient online robot learning. We explore algorithms which span a range of online learning paradigms, from online imitation learning, reinforcement learning, and bandit exploration and show how exploration can be made more efficient and safe without imposing significant additional supervision burden. We then show how these ideas can be used to enable safe and efficient robot learning for a number of different manipulation tasks in the real world such as robot knot tying, cable routing, and grasping.

The core idea in all of the algorithms in this dissertation is leveraging data from either offline datasets of desired or undesirable behaviors, or sparing human queries, to provide additional feedback to online robot learning algorithms to guide them towards task relevant states and away from constraint violating regions. To do this, we first study how sparing online human feedback can be used to accelerate learning new tasks while limiting the amount of burden imposed on human supervisors. We then study how offline data can be used to structure online exploration by either (1) identifying subsets of the environment in which to restrict exploration or (2) learn priors which bias exploration towards high performing actions.

1.1 Efficient Online Imitation Learning

We begin by exploring algorithms for learning from demonstrations, or imitation learning, which has been a very popular and successful approach for training robotic control policies in practice [8, 3, 9, 10]. Prior approaches for robotic imitation learning have largely focused on training policies using a set of offline expert demonstrations and then deploying them in the environment. However, this process can result in poor performance if the robot policy begins to visit states that differ sufficiently from those in the demonstrations, as there is no longer any reference behavior to imitate. This distribution shift has been shown to result in poor performance both in theory and practice [8]. To help bridge this gap, there has been significant interest in algorithms for online imitation learning, in which the robot has query access to an expert supervisor, which can take control of the robot or give various forms of feedback to the robot while the robot is executing its policy. While online access to a supervisor can significantly improve policy performance [8, 11, 12], it also comes at a cost, as supervisor queries, especially if the supervisor is a human, are often quite expensive.

In Chapters 2 and 3, we present two algorithms, LazyDAgger and ThriftyDAgger, which study how to sparingly request human feedback on a robot policy to strike a balance between achieving good policy performance and minimizing the burden imposed on the human supervisor to improve both the performance and scalability of online imitation learning algorithms for robotics. Then in Chapter 4, we study how online imitation learning algorithms can be used even without direct human supervision. Here, we explore how ideas from online imitation learning can be leveraged to iteratively distill model-based policies, which may be performant and sample efficient, but highly expensive to query, into reactive policies which can be rapidly evaluated while maintaining high quality performance.

1.2 Reinforcement Learning from Suboptimal Demonstrations

One key limitation of imitation learning algorithms is their inability to explore new, possibly higher performing behaviors than those provided by the expert supervisor. In practice, providing truly optimal demonstrations of a task can be challenging due to factors such as unintuitive robotic control interfaces or lack of human expertise in a task. To address this challenge, there has been significant interest in leveraging *suboptimal* demonstrations, which may be much more readily available in practice, to accelerate reinforcement learning. This simultaneously allows the robot to continually improve upon its behavior while giving it some initial signal about high performing behaviors.

In Chapters 5, 6, and 7 we present a series of three model-based reinforcement learning algorithms, Adjustable Boundary Condition LMPC (ABC-LMPC), Safety Augmented Value Estimation from Demonstrations (SAVED), and Latent Space Safe Sets (LS^3), which use suboptimal demonstrations to learn what we call a *safe set*, which measures the support of states from which the robot is confident in its ability to complete a task. As the robot samples more successful trajectories during online reinforcement learning, this safe set is iteratively expanded to capture the new states in the trajectories. The key idea in these three algorithms is to use this safe set to restrict exploration to the neighborhood of prior successful trajectories. This allows for stable and highly efficient learning, even in long horizon tasks with sparse rewards, as the robot is able to direct its exploration towards known promising regions in the state space, and does not spend time exploring behaviors that are very different from those which were previously successful. Then in Chapter 8, we study model-free reinforcement learning algorithms, and present Monte Carlo augmented Actor-Critic, a simple modification to the standard Bellman backup used in value-based reinforcement learning which leads to significantly faster learning from demonstrations.

1.3 Reinforcement Learning from Negative Demonstrations

One significant challenge with using reinforcement learning algorithms in practical robotic settings is that learning new tasks requires exploring a large space of possible behaviors, but in practice there may be certain behaviors or states which are known to be clearly suboptimal, dangerous, or generally unsafe/undesirable. Enabling safe exploration in reinforcement learning is challenging for two primary reasons: (1) to know how to be safe, you need to have some idea of what unsafe behaviors look like, and (2) there is an inherent conflict between exploring enough to learn some new task while restricting exploration in order to be safe. This motivates leveraging offline examples of unsafe behaviors to learn about such behaviors before online exploration, making it possible to learn new tasks without uncontrolled safety violations during deployment.

In Chapter 9, we present Recovery RL, an algorithm which first leverages a set of offline data of unsafe behaviors to learn a safety critic, which estimates the probability of constraint violation given the current state and a proposed action. Then, Recovery RL decouples learning across two policies: a task policy, which only optimizes a task specific reward function, and a recovery policy, which pushes the agent away from constraint violating states. The recovery policy is activated when the estimated probability of constraint violation from the safety critic exceeds some threshold, resulting in a switching controller that helps strike a balance between task performance and constraint satisfaction. However, it may typically be challenging to collect a large set of initial unsafe behaviors in safety critical environments, but may be comparatively easier to collect such behaviors in other environments with similar dynamics (eg. computational simulations or instrumented setups). To this end, in Chapter 10, we present MEta-learning for Safe Adaptation (MESA), which introduces a meta-learning approach to utilize the unsafe behaviors from related environments to learn a safety critic which can be rapidly adapted to a safety critical target environment.

1.4 Learning Priors for Rapid Bandit-Based Grasp Exploration

Bandit-based exploration algorithms have been one of the most successful classes of online learning algorithms in practical applications. They have been applied widely in web-based recommendation systems in online retail systems such as Amazon, entertainment such as YouTube and Netflix, and social media platforms such as Twitter and Facebook. However, many of the same challenges which make it difficult to use reinforcement learning algorithms for robotics make it challenging to use bandits. In the multi-armed bandit setting, the objective is to identify arms (actions) which correspond to high average reward, but selecting a given arm does not lead to a transition which changes the internal state of the system. This makes exploration somewhat less challenging, since there is no need to reason about long horizon plans or system dynamics. However, when there are sufficiently many actions to consider, exploration still takes significant time. This is particularly a challenge in robotics, in which efficient exploration is critical for algorithms to be useful on physical hardware.

We explore applying bandit exploration algorithms to robot grasping, where there is often a large number of possible actions (grasp candidate) to consider when identifying performant grasps. To accelerate exploration, we present Thompson Sampling with Learned Priors (TSLP) and Bandits for Online Rapid Grasp Exploration Strategy (BORGES) in Chapters 11 and 12 to learn a data-driven prior from general purpose grasping systems, which can give initial signal about which grasps may be promising and which grasps are clearly suboptimal. This prior can be readily computed from released, pre-trained grasping models, and interfaced directly with existing multi-armed bandit algorithms. This process allows these algorithms to rapidly refine this prior, making it feasible to quickly identify high quality grasps from up to 100 candidate grasps on a physical robot.

Part I

Efficient Online Imitation Learning

Chapter 2

LazyDAgger: Reducing Context Switching in Interactive Imitation Learning

Imitation learning allows a robot to learn from human feedback and examples [13, 14, 15]. In particular, *interactive imitation learning* (IL) [16, 12, 11], 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 [16]. However, a key challenge in interactive imitation learning is to reduce the burden that interventions place on the human supervisor [11, 12].

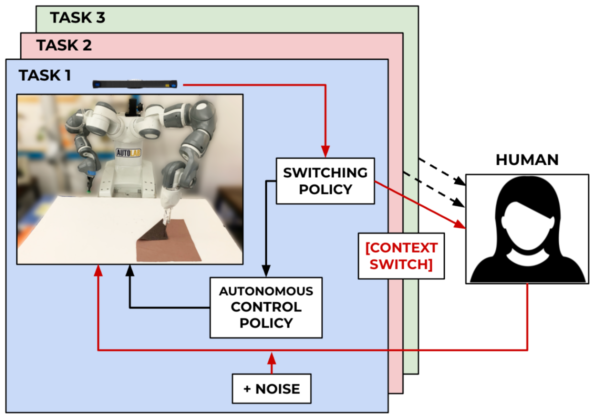


Figure 2.1: LazyDAgger learns to cede control to a supervisor in states in which it estimates that its actions will significantly deviate from those of the supervisor. LazyDAgger reduces context switches between supervisor and autonomous control to reduce burden on a human supervisor working on multiple 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 [17] or gain enough situational awareness to provide useful interventions [18].

We present LazyDAgger (Figure 2.1), 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 [19], where expressions are evaluated only when required to reduce computational burden. As in SafeDAgger [11], 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 [8] 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.

2.1 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 [13, 15, 14] 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 [20, 21, 22, 23], while others use reinforcement learning to train a policy using a reward function inferred from demonstrations [24, 25, 26, 27, 28]. Recent work has augmented demonstrations with additional offline information such as pairwise preferences [29, 30], human gaze [31], and natural language descriptions [32]. 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 [8].

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 [8]. 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 [33]. More generally, the idea of learning from action advice has been widely explored in imitation learning algorithms [34, 35, 36, 37]. There has also been significant recent interest in active preference queries for learning reward functions from pairwise preferences over demonstrations [38, 39, 40, 41, 42, 30]. 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 [16, 43].

Learning from Supervisor Interventions: There has been significant prior work on algorithms for learning policies from interventions. Xie et al. [44] and Kurenkov et al. [45] 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. [46], Nolan Wagener [47], and Saunders et al. [48] also leverage interventions from algorithmic policies, but for constraint satisfaction during learning. Kelly et al. [12], Spencer et al. [16], Wang et al. [49], Kahn, Abbeel, and Levine [50], Mandlekar et al. [51], and Amir et al. [52] 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 [53]. 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 [54, 55, 17, 56]. 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 and Cho [11] and Menda, Driggs-Campbell, and Kochenderfer [57] 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.

2.2 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 [58]. 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):

$$J(\pi_R) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi_R}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right], \tag{2.1}$$

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:

$$B(\pi) = L \cdot C(\pi) + D(\pi), \tag{2.2}$$

where L is context switch latency (Section 2) 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:

$$\pi = \arg\min_{\pi' \in \Pi} \{ J(\pi'_R) \mid B(\pi') \leq \Gamma_b \}, \tag{2.3}$$

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

2.3 Preliminaries: SafeDAgger

We consider interactive IL in the context of the objective introduced in Equation (2.3): to maximize task reward while limiting supervisor burden. To do this, LazyDAgger builds on SafeDAgger [11], 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:

$$\begin{aligned} \mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f) = &-f^*(\pi_R(s_t), \pi_H(s_t)) \log f(s_t) \\ &-(1 - f^*(\pi_R(s_t), \pi_H(s_t))) \log(1 - f(s_t)), \end{aligned} \tag{2.4}$$

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:

$$\pi(s_t) = \begin{cases} \pi_R(s_t) \text{ if } f(s_t) < 0.5 \\ \pi_H(s_t) \text{ otherwise,} \end{cases} \tag{2.5}$$

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.

2.4 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.

2.4.1 Action Discrepancy Prediction

SafeDAgger uses the classifier f to select between π_R and π_H (Equation (2.5)). However, in practice, this often leads to frequent context switching (Figure 2.3). 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 back to autonomous mode when the action discrepancy falls below a threshold β_R, where $\beta_R < \beta_H$. As illustrated in Figure 2.2, LazyDAgger's asymmetric switching criteria create a hysteresis band, as is often utilized in control theory [59]. Motivated by Eq. (2.3), we

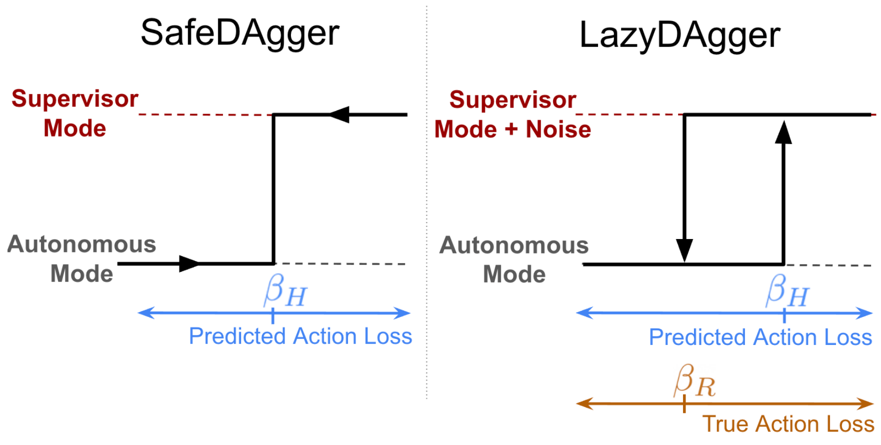


Figure 2.2: **LazyDAgger Switching Strategy:** SafeDAgger switches between supervisor and autonomous mode if the predicted action discrepancy is above threshold β_H. In contrast, LazyDAgger uses asymmetric switching criteria and switches to autonomous mode based on ground truth action discrepancy. The gap between β_R and β_H defines a hysteresis band [59].

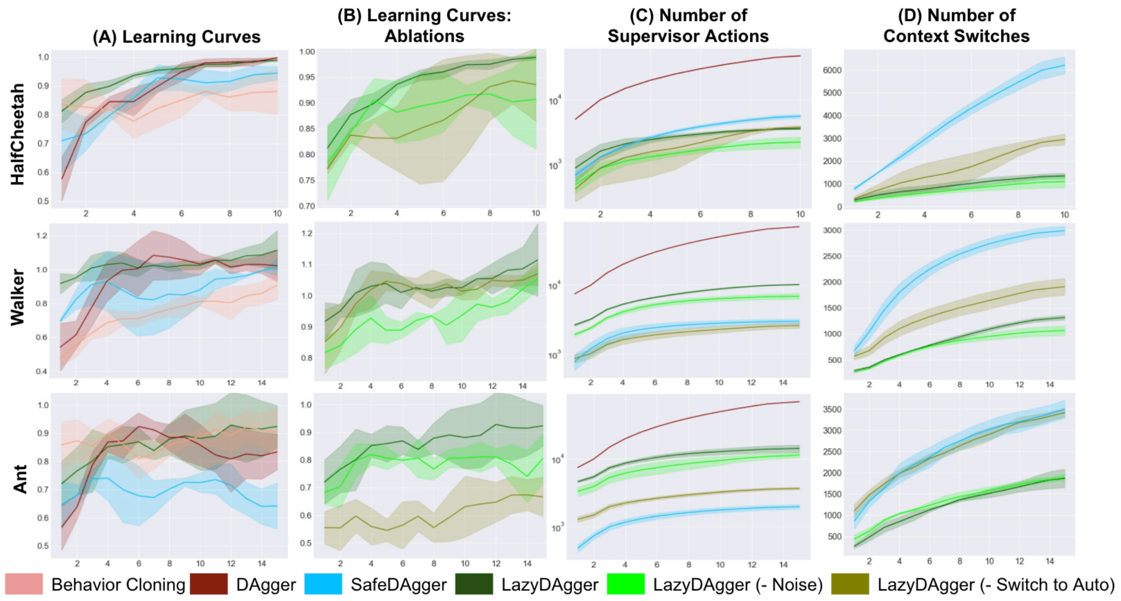


Figure 2.3: **MuJoCo Simulation Results:** We study task performance (A), ablations (B), online supervisor burden (C), and total bidirectional context switches (D) for LazyDAgger and baselines over 3 random seeds. For Columns (A)-(D), the x-axis for all plots shows the number of epochs over the training 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.

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.

2.4.2 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 discrepancy from the supervisor. Inspired by prior work that has identified noise injection as

Algorithm 1 The Student-Teacher Framework

```
Require: Collect $\mathcal{D}, \mathcal{D}_S$ offline with supervisor policy $\pi_H$
 1: $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}}\left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t))\right]$ {Eq. (2.1)}
 2: $f \leftarrow \arg\min_f \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}\cup\mathcal{D}_S}\left[\mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f)\right]$ {Eq. (2.4)}
 3: for $i \in \{1, \dots N\}$ do
 4:   Initialize $s_0$, Mode $\leftarrow$ Autonomous
 5:   for $t \in \{1, \dots T\}$ do
 6:     $a_t \sim \pi_R(s_t)$
 7:     if Mode = Supervisor or $f(s_t) \geq 0.5$ then
 8:       $a_t^H = \pi_H(s_t)$
 9:       $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t^H)\}$
10:       Execute $\tilde{a}_t^H \sim \mathcal{N}(a_t^H, \sigma^2 I)$
11:       if $\mathcal{L}(a_t, a_t^H) < \beta_R$ then
12:         Mode $\leftarrow$ Autonomous
13:       else
14:         Mode $\leftarrow$ Supervisor
15:       end if
16:     else
17:       Execute $a_t$
18:     end if
19:   end for
20:   $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}}\left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t))\right]$
21:   $f \leftarrow \arg\min_f \mathbb{E}_{(s_t,\pi_H(s_t))\sim\mathcal{D}\cup\mathcal{D}_S}\left[\mathcal{L}_S(\pi_R(s_t), \pi_H(s_t), f)\right]$
22: end for
```

a useful tool for improving the performance of imitation learning algorithms (e.g. Laskey et al. [33] and Brown, Goo, and Niekum [27]), 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 10 in Algorithm 1).

2.5 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.

2.5.1 Simulation Experiments: MuJoCo Benchmarks

Environments: We evaluate LazyDAgger and baselines on 3 continuous control environments from MuJoCo [61], a standard simulator for evaluating imitation and reinforcement

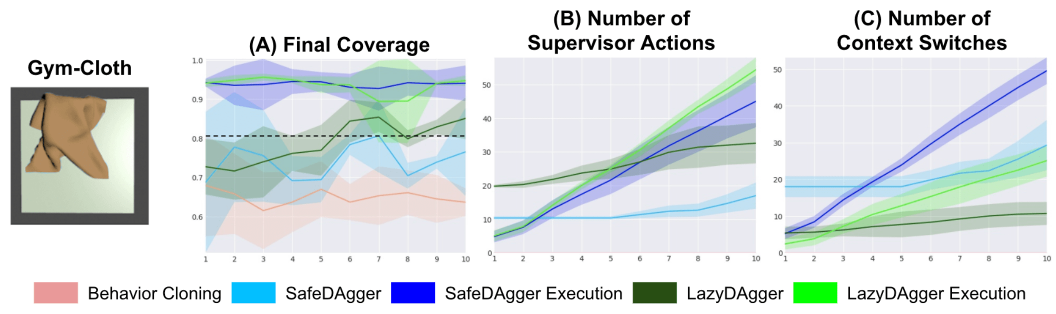


Figure 2.4: **Fabric Smoothing Simulation Results:** We study task performance measured by final fabric coverage (A), total supervisor actions (B), and total context switches (C) for LazyDAgger and baselines in the Gym-Cloth environment from [60]. The horizontal dotted line shows the success threshold 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.

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. (2.2)). 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 [23], DAgger [8], and SafeDAgger [11] 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 [62] 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 2.3, 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 2.3. Results suggest that LazyDAgger is able to match or outperform all baselines in terms of task performance across all simulation environments (Figure 2.3A). Additionally, LazyDAgger requires far fewer context switches compared to SafeDAgger (Figure 2.3D), while requesting a similar number of supervisor actions across domains (Figure 2.3C): 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 (2.2) 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 2.3), 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 2.1: **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 2.1): (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.

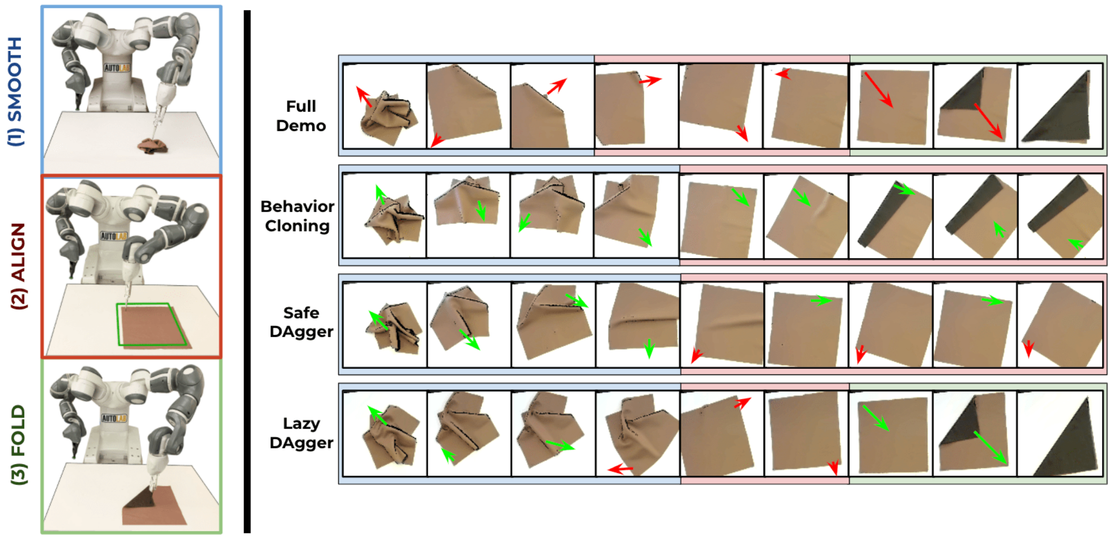


Figure 2.5: **Physical Fabric Manipulation Task:** *Left:* We evaluate on a 3-stage fabric manipulation task consisting of smoothing a crumpled fabric, aligning the fabric so all corners are visible in the observations, and performing a triangular fold. *Right:* Rollouts of the fabric manipulation task, 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.

2.5.2 Fabric Smoothing in Simulation

Environment: We evaluate LazyDAgger on the fabric smoothing task from [60] (shown in Figure 2.4) using the simulation environment from [60]. 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 [60], 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 [60] 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 [60]. 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 [60], 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 2.4. Figure 2.4 (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 2.4 (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 2.5.1) 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 2.5.3). We see that in this case, LazyDAgger achieves similar final coverage as SafeDAgger with significantly fewer context switches.

2.5.3 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 2.5). 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. [63]; see Figure 2.5), 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 2.1, 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 2.5 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 2.5). 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 2.1), 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.

2.6 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.

Chapter 3

ThriftyDAgger: Budget-Aware Novelty and Risk Gating for Interactive Imitation Learning

Imitation learning (IL) [13, 10, 15] has seen success in a variety of robotic tasks ranging from autonomous driving [20, 64, 65] to robotic manipulation [66, 67, 68, 69, 70]. In its simplest form, the human provides an offline set of task demonstrations to the robot, which the robot uses to match human behavior. However, this offline approach can lead to low task performance due to a mismatch between the state distribution encountered in the demonstrations and that visited by the robot [8, 33], resulting in brittle policies that cannot be effectively deployed in real-world applications [71]. *Interactive imitation learning*, in which the robot periodically cedes control to a human supervisor for corrective interventions, has emerged as a promising technique to address these challenges [12, 16, 37, 72]. However, while interventions make it possible to learn robust policies, these interventions require significant human time. Thus, the central challenge in interactive IL algorithms is to control the timing and length of interventions to balance task performance with the burden imposed on the human supervisor [11, 72]. Achieving this balance is even more critical if the human supervisor must oversee multiple robots at once [54, 73, 17], for instance supervising a fleet of self-driving taxis [65] or robots in a warehouse [74]. Since even relatively reliable robot policies inevitably encounter new situations that must fall back on human expertise, this problem is immediately relevant to contemporary companies such as Waymo and Plus One Robotics.

One way to determine when to solicit interventions is to allow the human supervisor to decide when to provide the corrective interventions. However, these approaches—termed “human-gated” interactive IL algorithms [12, 16, 51]—require the human supervisor to continuously monitor the robot to determine when to intervene. This imposes significant burden on the supervisor and cannot effectively scale to settings in which a small number of humans supervise a large number of robots. To address this challenge, there has been recent interest in approaches that enable the robot to actively query humans for interventions, called

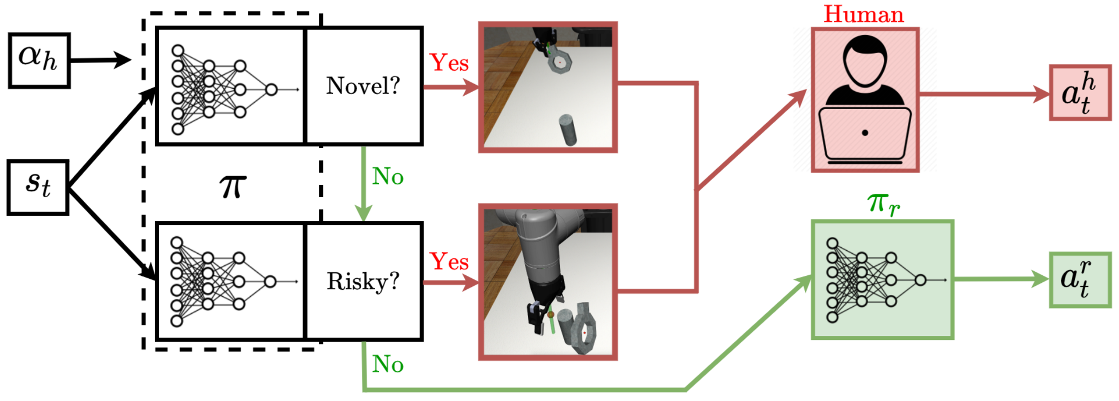


Figure 3.1: **ThriftyDAgger:** Given a desired context switching rate α_H, ThriftyDAgger transfers control to a human supervisor if the current state s_t is (1) sufficiently novel or (2) sufficiently risky, indicating that the probability of task success is low under robot policy π_R. Intuitively, one should not only distrust π_R in states significantly out of the distribution of previously encountered states, but should also cede control to a human supervisor in more familiar states where the robot predicts that it is unlikely to successfully complete the task.

"robot-gated" algorithms [11, 35, 57, 72]. Robot-gated methods allow the robot to reduce burden on the human supervisor by only requesting interventions when necessary, switching between robot control and human control based on some intervention criteria. Hoque et al. [72] formalize the idea of supervisor burden as the expected total cost incurred by the human in providing interventions, which consists of the expected cost due to *context switching* between autonomous and human control and the time spent actually providing interventions. However, it is difficult to design intervention criteria that limit this burden while ensuring that the robot gains sufficient information to imitate the supervisor's policy.

This chapter makes several contributions. First, we develop intervention criteria based on a synthesis of two estimated properties of a given state: *novelty*, which measures whether the state is significantly out of the distribution of previously encountered states, indicating that the robot policy should not be trusted; and *risk*, which measures the likelihood of the robot successfully completing the task on its own. While state novelty has been considered in prior work [57], the key insight in our intervention criteria lies in combining novelty with a new risk metric to estimate the probability of task success. Second, we present a new robot-gated interactive IL algorithm, ThriftyDAgger (Figure 3.1), which employs these measures jointly to solicit human interventions only when necessary. Third, while prior robot-gated algorithms [11, 72] require careful parameter tuning to modulate the timing and frequency of human intervention requests, ThriftyDAgger only requires the supervisor to specify a desired context switching rate and sets thresholds accordingly. Fourth, experimental results demonstrate ThriftyDAgger's effectiveness for reducing supervisor burden while learning challenging tasks both in simulation and in an image-based cable routing task on a physical robot. Finally, the results of a human user study applying ThriftyDAgger to control a fleet of three simulated robots suggest that ThriftyDAgger significantly improves performance on both the robots' task and an independent human task while imposing fewer context switches, fewer human intervention actions, and lower mental load and frustration than prior algorithms.

3.1 Related Work

Imitation Learning from Human Feedback: There has been significant prior work in offline imitation learning, in which the agent leverages an offline dataset of expert demonstrations either to directly match the distribution of trajectories in the offline dataset [20, 26, 13, 15, 14, 21, 22], for instance via Behavior Cloning [23, 75], or to learn a reward function that can then be optimized via reinforcement learning [24, 26, 27]. However, while these approaches have shown significant success in a number of domains [66, 69, 68, 75], learning from purely offline data leads to a trajectory distribution mismatch which yields suboptimal performance both in theory and practice [8, 33]. To address this problem, there have been a number of approaches that utilize online human feedback while the agent acts in the environment, such as providing suggested actions [8, 34, 36, 37] or preferences [38, 39, 40, 41, 42, 30]. However, many of these forms of human feedback may be unreliable if the robot visits states that significantly differ from those the human supervisor would themselves visit; in such situations, it is challenging for the supervisor to determine what correct behavior should look like without directly interacting with the environment [16, 43].

Interactive Imitation Learning: A natural way to collect reliable online feedback for imitation learning is to periodically cede control to a human supervisor, who then provides a corrective intervention to illustrate desired behavior. Human-gated interactive IL algorithms [12, 16, 51] such as HG-DAgger [12] require the human to determine when to engage in interventions. However, these algorithms require a human to continuously monitor the robot to determine when to intervene, which imposes significant burden on the supervisor and is particularly impractical if a small number of humans must supervise a large number of robots. Furthermore, it requires the human to determine when the robot needs help and when to cede control, which can be unintuitive and unreliable.

By contrast, robot-gated interactive IL algorithms, such as EnsembleDAgger [57], SafeDAgger [11], and LazyDAgger [72], allow the robot to actively query for human interventions. In practice, these algorithms estimate various quantities correlated with task performance [11, 72, 76, 35] and uncertainty [57] and use them to determine when to solicit interventions. Prior work has proposed intervention criteria which use the novelty of states visited by the robot [57] or the predicted discrepancy between the actions proposed by the robot policy and those of the supervisor [11, 72]. However, while state novelty provides a valuable signal for soliciting interventions, we argue that this alone is insufficient, as a state's novelty does not convey information about the level of precision with which actions must be executed in that state. In practice, many robotic tasks involve moving through critical "bottlenecks" [51], which, though not necessarily novel, still present challenges. Examples include moving an eating utensil close to a person's mouth or placing an object on a shelf without disturbing nearby objects. Similarly, even if predicted accurately, action discrepancy is often a flawed risk measure, as high action discrepancy between the robot and the supervisor may be permissible when fine-grained control is not necessary (e.g. a robot gripper moving in free space) but impermissible when precision is critical (e.g. a robot gripper actively trying to grasp an object). In contrast, ThriftyDAgger presents an intervention criteria incorporating both state novelty and a novel risk metric and automatically tunes key parameters, allowing efficient use of human supervision.

3.2 Problem Statement

Given a robot, a task for the robot to accomplish, and a human supervisor with a specified context switching budget, the goal is to train the robot to imitate supervisor performance within the budget. We model the robot environment as a discrete-time Markov Decision Process (MDP) $\mathcal{M}$ with continuous states $s \in \mathcal{S}$, continuous actions $a \in \mathcal{A}$, and time horizon T [58]. We consider the interactive imitation learning (IL) setting [12], where the robot does not have access to a shaped reward function or to the MDP's transition dynamics but can temporarily cede control to a supervisor who uses policy $\pi_H : \mathcal{S} \to \mathcal{A}$. We specifically focus on tasks where there is a goal set $\mathcal{G}$ which determines success, but that can be challenging and long-horizon, making direct application of RL highly sample inefficient.

We assume that the human and robot utilize the same action space (e.g. through a teleoperation interface) and that task success can be specified by convergence to some goal set $\mathcal{G} \subseteq \mathcal{S}$ within the time horizon (i.e., the task is successful if $\mathcal{G}$ is reached within T timesteps). We further assume access to an indicator function $\mathbb{1}_{\mathcal{G}} : \mathcal{S} \to \{0, 1\}$, which indicates whether a state belongs to the goal set $\mathcal{G}$.

The IL objective is to minimize a surrogate loss function $J(\pi_R)$ to encourage the robot policy $\pi_R : \mathcal{S} \to \mathcal{A}$ to match π_H:

$$J(\pi_R) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi_R}} \left[\mathcal{L}(\pi_R(s_t), \pi_H(s_t)) \right], \tag{3.1}$$

where $\mathcal{L}(\pi_R(s), \pi_H(s))$ is an action discrepancy measure between $\pi_R(s)$ and $\pi_H(s)$ (e.g. MSE loss), and $d_t^{\pi_R}$ is the marginal state distribution at timestep t induced by the robot policy π_R in $\mathcal{M}$.

In the interactive IL setting, in addition to optimizing Equation (3.1), a key design goal is to minimize the imposed burden on the human supervisor. To formalize this, we define a switching policy π, which determines whether the system is under robot control π_R (which we call *autonomous mode*) or human supervisor control π_H (which we call *supervisor mode*). Following prior work [72], we define $C(\pi)$, the expected number of *context switches* in an episode under policy π, as follows: $C(\pi) = \sum_{t=1}^{T} \mathbb{E}_{s_t \sim d_t^{\pi}} \left[m_I(s_t; \pi) \right]$, where $m_I(s_t; \pi)$ is an indicator for whether or not a context switch occurs from autonomous to supervisor control. Similarly, we define $I(\pi)$ as the expected number of *supervisor actions* in an intervention solicited by π. We then define the total burden $B(\pi)$ imposed on the human supervisor as follows:

$$B(\pi) = C(\pi) \cdot \big(L + I(\pi)\big), \tag{3.2}$$

where L is the *latency* of a context switch between control modes (summed over both switching directions) in units of timesteps, where each action takes one timestep. The interactive IL objective is to minimize the discrepancy from the supervisor policy while limiting supervisor burden within some Γ_b:

$$\pi = \arg\min_{\pi' \in \Pi} \{J(\pi_R) \mid B(\pi') \leq \Gamma_b\}. \tag{3.3}$$

Because it is challenging to explicitly optimize policies to satisfy the supervisor burden constraint in Equation (3.3), we present novel intervention criteria that enable reduction of supervisor burden by limiting the total number of interventions to a user-specified budget. Given sufficiently high latency L, limiting the interventions $C(\pi)$ directly corresponds to limiting supervisor burden $B(\pi)$.

3.3 ThriftyDAgger

ThriftyDAgger determines when to switch between autonomous and human supervisor control modes by leveraging estimates of both the *novelty* and *risk* of states. Below, Sections 3.3.1 and 3.3.2 discuss the estimation of state novelty and risk of task failure, respectively, while Section 3.3.3 discusses ThriftyDAgger's integration of these measures to determine when to switch control modes. Section 3.3.4 then describes an online procedure to set thresholds for switching between control modes. Finally, Section 3.3.5 describes the full control flow of ThriftyDAgger.

3.3.1 Novelty Estimation

When the robot policy visits states that lie significantly outside the distribution of those encountered in the supervisor trajectories, it does not have any reference behavior to imitate. This motivates initiating interventions to illustrate desired recovery behaviors in these states. However, estimating the support of the state distribution visited by the human supervisor is challenging in the high-dimensional state spaces common in robotics. Following prior work [57], we train an ensemble of policies with bootstrapped samples of transitions from supervisor trajectories. We then measure the novelty of a given state s by calculating the variance of the policy outputs at state s across ensemble members. In practice, the action $a \in \mathcal{A}$ outputted by each policy is a vector; thus, we measure state novelty by computing the variance of each component of the action vector a across the ensemble members and then averaging over the components. We denote this quantity by Novelty(s). Once in supervisor mode, as noted in Hoque et al. [72], we can obtain a more precise correlate of novelty by computing the ground truth action discrepancy between the supervisor's actions and those of the robot policy.

3.3.2 Risk Estimation

Interventions may be required not only in novel states outside the distribution of supervisor trajectories, but also in familiar states that are prone to result in task failure. For example, a task might have a "bottleneck" region with low tolerance for error, which has low novelty but nevertheless requires more supervision to learn a reliable robot policy. To address this challenge, we propose a novel measure of a state's "riskiness," capturing the likelihood that the robot cannot successfully converge to the goal set $\mathcal{G}$. We first define a Q-function to quantify the discounted probability of successful convergence to $\mathcal{G}$ from a given state and action under the robot policy:

$$Q_{\mathcal{G}}^{\pi_R}(s_t, a_t) = \mathbb{E}_{s_{t'} \sim d_{t'}^{\pi_R}} \left[\sum_{t'=t}^{\infty} \gamma^{t'-t} \mathbb{1}_{\mathcal{G}}(s_{t'}) | s_t, a_t \right], \tag{3.4}$$

where $\mathbb{1}_{\mathcal{G}}(s_t)$ is equal to 1 if s_t belongs to $\mathcal{G}$. We estimate $Q_{\mathcal{G}}^{\pi_R}(s_t, a_t)$ via a function approximator $\hat{Q}_{\phi,\mathcal{G}}^{\pi_R}$ parameterized by ϕ, and define a state's riskiness in terms of this learned Q-function:

$$\text{Risk}^{\pi_R}(s, a) = 1 - \hat{Q}_{\phi,\mathcal{G}}^{\pi_R}(s, a). \tag{3.5}$$

In practice, we train $\hat{Q}_{\phi,\mathcal{G}}^{\pi_R}$ on transitions (s_t, a_t, s_{t+1}) from both autonomous mode and supervisor mode by minimizing the following MSE loss inspired by [46]:

$$J_{\mathcal{G}}^{Q}(s_t, a_t, s_{t+1}; \phi) = \frac{1}{2} \left(\hat{Q}_{\phi,\mathcal{G}}^{\pi_R}(s_t, a_t) - (\mathbb{1}_{\mathcal{G}}(s_t) + (1 - \mathbb{1}_{\mathcal{G}}(s_t)) \gamma \hat{Q}_{\phi,\mathcal{G}}^{\pi_R}(s_{t+1}, \pi_R(s_{t+1}))) \right)^2. \tag{3.6}$$

Note that since $\hat{Q}_{\phi,\mathcal{G}}^{\pi_R}$ is only used to solicit interventions, it must only be accurate enough to distinguish risky states from others, rather than be able to make the fine-grained distinctions between different states required for accurate policy learning in reinforcement learning.

3.3.3 Regulating Switches in Control Modes

We now describe how ThriftyDAgger leverages the novelty estimator from Section 3.3.1 and the risk estimator from Section 3.3.2 to regulate switches between autonomous and supervisor control. While in autonomous mode, the switching policy π initiates a switch to supervisor mode at timestep t if either (1) state s_t is sufficiently unfamiliar or (2) the robot policy has a low probability of task success from s_t. Stated precisely, π initiates a switch to supervisor mode from autonomous mode at timestep t if the predicate Intervene(s_t, δ_H, β_H) evaluates to

TRUE, where Intervene(s_t, δ_H, β_H) is TRUE if (1) Novelty$(s_t) > \delta_H$ or (2) $\text{Risk}^{\pi_R}(s_t, \pi_R(s_t)) > \beta_H$, and FALSE otherwise. Note that the proposed switching policy only depends on Risk^{π_R} for states which are *not* novel (as novel states already initiate switches to supervisor control regardless of risk), since the learned risk measure should only be trusted on states in the neighborhood of those on which it has been trained.

In supervisor mode, π switches to autonomous mode if the action discrepancy between the human and robot policy and the robot's task failure risk are both below threshold values (Section 3.3.4), indicating that the robot is in a familiar and safe region. Stated precisely, π switches to autonomous mode from supervisor mode if the predicate Cede(s_t, δ_R, β_R) evaluates to TRUE, where Cede(s_t, δ_R, β_R) is TRUE if (1) $||\pi_R(s_t) - \pi_H(s_t)||_2^2 < \delta_R$ and (2) $\text{Risk}^{\pi_R}(s_t, \pi_R(s_t)) < \beta_R$, and FALSE otherwise. Here, the risk metric ensures that the robot has a high probability of autonomously completing the task, while the coarser 1-step action discrepancy metric verifies that we are in a familiar region of the state space where the $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ values can be trusted. Motivated by prior work [72] and hysteresis control [59], we use asymmetric switching criteria with stricter thresholds in supervisor mode ($\beta_R < \beta_H$) to encourage lengthier interventions and reduce context switches experienced by the human supervisor.

3.3.4 Computing Risk and Novelty Thresholds from Data

One challenge of the control strategy presented in Section 3.3.3 lies in tuning the key parameters $(\delta_H, \delta_R, \beta_H, \beta_R)$ governing when context switching occurs. As noted in prior work [57], performance and supervisor burden can be sensitive to these thresholds. To address this difficulty, we assume that the user specifies their availability in the form of a desired intervention budget $\alpha_H \in [0, 1]$, indicating the desired proportion of timesteps in which interventions will be requested. This desired context switching rate can be interpreted in the context of supervisor burden as defined in Equation (3.2): if the latency of a context switch dominates the time cost of the intervention itself, limiting the expected number of context switches to within some intervention budget directly limits supervisor burden.

Given α_H, we set β_H to be the $(1 - \alpha_H)$-quantile of $\text{Risk}^{\pi_R}(s, \pi_R(s))$ for all states previously visited by π_R and set δ_H to be the $(1 - \alpha_H)$-quantile of Novelty(s) for all states previously visited by π_R. We set δ_R to be the mean action discrepancy on the states visited by the supervisor after π_R is trained and set β_R to be the median of $\text{Risk}^{\pi_R}(s, \pi_R(s))$ for all states previously visited by π_R. (Note that β_R can easily be set to different quantiles to adjust mean intervention length if desired.) We find that these settings strike a balance between informative interventions and imposed supervisor burden.

3.3.5 ThriftyDAgger Overview

We now summarize the ThriftyDAgger procedure, with full pseudocode available in the supplement. ThriftyDAgger first initializes π_R via Behavior Cloning on offline transitions ($\mathcal{D}_H$ from the human supervisor, π_H). Then, π_R collects an initial offline dataset $\mathcal{D}_R$ from the resulting π_R, initializes $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ by optimizing Equation (3.5) on $\mathcal{D}_R \cup \mathcal{D}_H$, and initializes parameters $\beta_H, \beta_R, \delta_H$, and δ_R as in Section 3.3.4. We then collect data for N episodes, each with up to T timesteps. In each timestep of each episode, we determine whether robot policy π_R or human supervisor π_H should be in control using the procedure in Section 3.3.3. Transitions in autonomous mode are aggregated into $\mathcal{D}_R$ while transitions in supervisor mode are aggregated into $\mathcal{D}_H$. After each episode, π_R is updated via supervised learning on $\mathcal{D}_H$, and $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ is then updated on $\mathcal{D}_R \cup \mathcal{D}_H$ to reflect the probability of task success of the updated π_R.

3.4 Experiments

In the following experiments, we study whether ThriftyDAgger can balance task performance and supervisor burden more effectively than prior IL algorithms in three contexts: (1) training a simulated robot to perform a peg insertion task (Section 3.4.3); (2) supervising a fleet of three simulated robots to perform the peg insertion task in a human user study (Section 3.4.4); and (3) training a physical surgical robot to perform a cable routing task (Section 3.4.5). In the supplementary material, we also include results from an additional simulation experiment on a challenging block stacking task.

3.4.1 Evaluation Metrics

We consider ThriftyDAgger's performance during training and execution. For the latter, we evaluate both the (1) *autonomous success rate*, or success rate when deployed after training without access to a human supervisor, and (2) *intervention-aided success rate*, or success rate when deployed after training with a human supervisor in the loop. These metrics are reported in the Peg Insertion study (Section 3.4.3) and the Physical Cable Routing study (Section 3.4.5). For all experiments, during both training and intervention-aided execution, we evaluate the number of interventions, human actions, and robot actions per episode. These metrics are computed over successful episodes only to prevent biasing the metrics by the maximum episode horizon length T; such bias occurs, for instance, when less successful policies appear to take more actions due to hitting the time boundary more often. Additional metrics including cumulative statistics across all episodes are reported in the supplement.

In our user study (Section 3.4.4), we also report the following quantities: throughput (total number of task successes across the three robots), performance on an independent human task, the idle time of the robots in the fleet, and users' qualitative ratings of mental load and frustration. By comparing the amount of human supervision and success rates across different algorithms, we are interested in evaluating how effectively each algorithm balances supervision with policy performance.

3.4.2 Comparisons

We compare ThriftyDAgger to the following algorithms: Behavior Cloning, which does not use interventions; HG-DAgger [12], which is human-gated and always requires supervision; SafeDAgger [11], which is robot-gated and performs interventions based on estimated action discrepancy between the human supervisor and robot policy; and LazyDAgger [72], which builds on SafeDAgger by introducing asymmetric switching criteria to encourage lengthier interventions. We also implement two ablations: one that does not use a novelty measure to regulate context switches (ThriftyDAgger (-Novelty)) and one that does not use risk to regulate context switches (ThriftyDAgger (-Risk)).

3.4.3 Peg Insertion in Simulation

We first evaluate ThriftyDAgger on a long-horizon (100+ timesteps) peg insertion task (Figure 3.2) from the Robosuite simulation environment [77]. The goal is to grasp a ring in a random initial pose and thread it over a cylinder at a fixed target location. This task has two bottlenecks which motivate learning from interventions: (1) correctly grasping the ring and (2) correctly placing it over the cylinder. A human teleoperates the robot through a keyboard interface to provide interventions. The states consist of the robot's joint angles and ring's pose, while the actions specify 3D translation, rotation, and opening or closing the gripper. For ThriftyDAgger and its ablations, we use target intervention frequency $\alpha_H = 0.01$ (Section 3.3.4). We collect 30 offline task demos (2,687 state-action pairs) from a human supervisor to initialize the robot policy for all compared algorithms. Behavior Cloning is given additional state-action pairs roughly equivalent to the average amount of supervisor actions solicited by the interactive algorithms (Table B.1 in the appendix). For ThriftyDAgger and each interactive IL baseline, we perform 10,000 environment steps, during which each episode takes at most 175 timesteps and system control switches between the human and robot. Hyperparameter settings for all algorithms are detailed in the supplement.

Results (Table 3.1) suggest that ThriftyDAgger simultaneously solicits fewer interventions and achieves a significantly higher autonomous success rate than prior robot-gated algorithms, although it does request more human actions due to its conservative exit criterion for interventions (Cede(s_t, δ_R, β_R)). The number of human actions falls significantly at execution time (Table 3.1), when the robot policy has been trained on online data and is therefore less risky. We find that all interactive IL algorithms substantially outperform Behavior Cloning, which does not have access to supervisor interventions. Notably, ThriftyDAgger achieves a higher autonomous success rate than even HG-DAgger, in which the supervisor is able to decide the timing and length of interventions. This indicates that ThriftyDAgger's intervention criteria enable it to autonomously solicit interventions as informative as those chosen by a human supervisor with expert knowledge of the task. Furthermore, ThriftyDAgger achieves a 100% intervention-aided success rate at execution time, suggesting that ThriftyDAgger successfully identifies the required states at which to solicit interventions. We find that both ablations of ThriftyDAgger (Ours (-Novelty) and Ours (-Risk)) achieve significantly lower

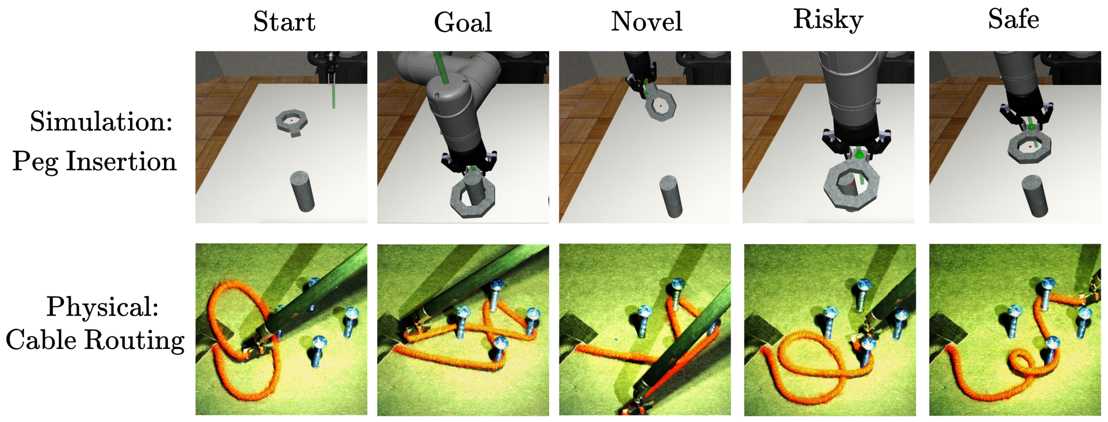


Figure 3.2: **Experimental Domains:** We visualize the peg insertion simulation domain (top row) and the cable routing domain with the physical robot (bottom row). We visualize sample start and goal states, in addition to states which ThriftyDAgger categorizes as novel, risky, or neither. ThriftyDAgger marks states as novel if they are far from states that the supervisor visited and risky if the robot is stuck in a bottleneck, e.g. if the ring is wedged against the side of the cylinder (top) or the cable is near all four obstacles (bottom).

autonomous success rates, indicating that both the novelty and risk measures are critical to ThriftyDAgger's performance. We calculate ThriftyDAgger's context switching rate to be 1.15% novelty switches and 0.79% risk switches, both approximately within the budget of $\alpha_H = 0.01$.

3.4.4 User Study: Controlling A Fleet of Three Robots in Simulation

We conduct a user study with 10 participants (7 male and 3 female, aged 18-37). Participants supervise a fleet of three simulated robots, each performing the peg insertion task from Section 3.4.3. We evaluate how different interactive IL algorithms affect the participants' (1) ability to provide effective robot interventions, (2) performance on a distractor task performed between robot interventions, and (3) levels of mental demand and frustration. For the distractor task, we use the game Concentration (also known as Memory or Matching Pairs), in which participants identify as many pairs of matching cards as possible among a set of face-down cards. This is intended to emulate tasks which require continual focus, such as cooking a meal or writing a research paper, in which frequent context switches between performing the task and helping the robots is frustrating and degrades performance.

The participants teleoperate the robots using three robot-gated interactive IL algorithms: SafeDAgger, LazyDAgger, and ThriftyDAgger. The participant is instructed to make progress on the distractor task only when no robot requests an intervention. When an intervention is requested, the participant is instructed to pause the distractor task, provide

Table 3.1: **Peg Insertion in Simulation Results:** We first report training performance (number of interventions (Ints), number of human actions (Acts (H)), and number of robot actions (Acts (R))) and report the success rate of the fully-trained policy at execution time when no interventions are allowed (Auto Succ.). We then evaluate the fully-trained policies with interventions allowed and report the same intervention statistics and the success rate (Int-Aided Succ.). We find that ThriftyDAgger achieves the highest autonomous and intervention-aided success rates among all algorithms compared. Notably, ThriftyDAgger even achieves a higher autonomous success rate than HG-DAgger, in which the human decides when to intervene during training.

Algorithm	Training Interventions			Auto Succ.	Execution Interventions			Int-Aided Succ.
	Ints	Acts (H)	Acts (R)		Ints	Acts (H)	Acts (R)	
Behavior Cloning	N/A	N/A	108.0 ± 15.9	24/100	N/A	N/A	N/A	N/A
SafeDAgger	3.89 ± 1.44	19.8 ± 9.9	88.8 ± 19.4	24/100	4.00 ± 1.37	19.5 ± 5.3	77.5 ± 11.7	17/20
LazyDAgger	1.46 ± 1.15	13.2 ± 12.4	102.1 ± 18.2	48/100	1.73 ± 1.29	12.6 ± 14.4	91.7 ± 24.0	11/20
HG-DAgger	1.49 ± 0.88	20.3 ± 15.6	97.1 ± 17.5	57/100	1.15 ± 0.73	17.1 ± 11.6	103.6 ± 14.0	**20/20**
Ours (-Novelty)	$\mathbf{0.79 \pm 0.81}$	35.1 ± 23.1	70.0 ± 35.8	49/100	$\mathbf{0.33 \pm 0.62}$	2.5 ± 5.0	114.0 ± 26.0	12/20
Ours (-Risk)	0.99 ± 0.96	7.8 ± 12.0	104.2 ± 19.2	49/100	1.39 ± 0.95	9.8 ± 12.0	109.1 ± 22.9	18/20
Ours: ThriftyDAgger	0.88 ± 1.01	43.6 ± 24.5	60.0 ± 32.8	**73/100**	1.35 ± 0.66	21.3 ± 15.0	84.8 ± 21.8	**20/20**

an intervention from the requested state until the robot (or multiple robots queued after each other) no longer requires assistance, and then return to the distractor task. The participants also teleoperate with HG-DAgger, where they no longer perform the distractor task and are instructed to continually monitor all three robots simultaneously and decide on the length and timing of interventions themselves. Each algorithm runs for 350 timesteps, where in each timestep, all robots in autonomous mode execute one action and the human executes one action on the currently supervised robot (if applicable). The supplement illustrates the user study interface and fully details the experiment protocol. All algorithms are initialized as in Section 3.4.3.

Results (Table 3.2) suggest that ThriftyDAgger achieves significantly higher throughput than all prior algorithms while requiring fewer interventions and fewer human actions, indicating that ThriftyDAgger requests interventions more judiciously than prior algorithms. Furthermore, ThriftyDAgger also enables a lower mean idle time for robots and higher performance on the distractor task. Notably, ThriftyDAgger solicits fewer interventions and total actions while achieving a higher throughput than HG-DAgger, in which the participant chooses when to intervene. We also report metrics of users' mental workload and frustration using the NASA-TLX scale [78] in the supplement. Results suggest that users experience lower degrees of frustration and mental load when interacting with ThriftyDAgger and LazyDAgger compared to HG-DAgger and SafeDAgger. We hypothesize that participants struggle with HG-DAgger due to the difficultly of monitoring multiple robots simultaneously, while SafeDAgger's frequent context switches lead to user frustration during experiments.

Table 3.2: **Three-Robot Fleet Control User Study Results:** Results for experiments with 10 human subjects and 3 simulated robots on the peg insertion task. We report the total numbers of interventions, human actions, and robot actions, as well as the throughput, or total task successes achieved across robots, for all algorithms. Additionally, for robot-gated algorithms, we report the Concentration score (number of pairs found) and the mean idle time of robots in the fleet in timesteps. Results suggest that ThriftyDAgger outperforms all prior algorithms across all metrics, requesting fewer interventions and total human actions while achieving higher throughput, lowering the robots' mean idle time, and enabling higher performance on the Concentration task.

Algorithm	Interventions	Human Actions	Robot Actions	Concentration Pairs	Throughput	Mean Idle Time
HG-DAgger	10.6 ± 2.5	198.0 ± 32.1	834.4 ± 38.1	N/A	5.1 ± 1.9	N/A
SafeDAgger	22.1 ± 4.8	234.1 ± 31.8	700.7 ± 70.4	17.7 ± 8.2	3.0 ± 2.4	38.4 ± 14.1
LazyDAgger	10.0 ± 2.1	219.5 ± 43.3	719.2 ± 89.7	20.9 ± 7.9	5.1 ± 1.7	37.1 ± 20.5
Ours: ThriftyDAgger	$\mathbf{7.9 \pm 2.1}$	$\mathbf{179.4 \pm 34.9}$	793.2 ± 86.6	$\mathbf{33.0 \pm 8.5}$	$\mathbf{9.2 \pm 2.0}$	$\mathbf{25.8 \pm 19.3}$

3.4.5 Physical Experiment: Visuomotor Cable Routing

Finally, we evaluate ThriftyDAgger on a long-horizon cable routing task with a da Vinci surgical robot [79]. Here, the objective is to route a red cable into a Figure-8 pattern around 4 pegs via teleoperation with the robot's master controllers (see supplement). The algorithm only observes high-dimensional 64×64×3 RGB images of the workspace and generates continuous actions representing delta-positions in (x, y). As in Section 3.4.3, ThriftyDAgger uses a target intervention frequency of $\alpha_H = 0.01$. We collect 25 offline task demonstrations (1,381 state-action pairs) from a human supervisor to initialize the robot policy for ThriftyDAgger and all comparisons. We perform 1,500 environment steps, where each episode has at most 100 timesteps and system control can switch between the human and robot. The supplement details the hyperparameter settings for all algorithms.

Results (Table 3.3) suggest that both ThriftyDAgger and HG-DAgger achieve a significantly higher autonomous success rate than Behavior Cloning, which is never able to complete the task. Furthermore, ThriftyDAgger achieves a comparable autonomous success rate to HG-DAgger while requesting fewer interventions and a similar number of total human actions. This again suggests that ThriftyDAgger's intervention criteria enable it to solicit interventions equally as informative or more informative than those chosen by a human supervisor. Finally, at execution time ThriftyDAgger achieves a 100% intervention-aided success rate with minimal supervision, again indicating that ThriftyDAgger successfully identifies the timing and length of interventions to increase policy reliability.

3.5 Discussion and Future Work

We present ThriftyDAgger, a scalable robot-gated interactive imitation learning algorithm that leverages learned estimates of state novelty and risk of task failure to reduce burden on

Table 3.3: **Physical Cable Routing Results:** We first report intervention statistics during training (number of interventions (Ints), number of human actions (Acts (H)), and number of robot actions (Acts (R))) and report the success rate of the fully-trained policy at execution time when no interventions are allowed (Auto Succ.). We then evaluate the fully-trained policies with interventions allowed and report the same intervention statistics and the success rate (Int-Aided Succ.). We find that ThriftyDAgger achieves the highest autonomous and intervention-aided success rates among all algorithms compared. Notably, ThriftyDAgger achieves a comparable autonomous success rate to HG-DAgger, in which the human decides when to intervene during training.

Algorithm	Training Interventions			Auto Succ.	Execution Interventions			Int-Aided Succ.
	Ints	Acts (H)	Acts (R)		Ints	Acts (H)	Acts (R)	
Behavior Cloning	N/A	N/A	N/A	0/15	N/A	N/A	N/A	N/A
HG-DAgger	1.55 ± 1.16	13.9 ± 10.9	55.5 ± 10.9	10/15	$\mathbf{0.40 \pm 0.49}$	2.7 ± 3.5	73.9 ± 7.9	**15/15**
Ours: ThriftyDAgger	$\mathbf{1.42 \pm 1.14}$	15.2 ± 12.4	45.5 ± 18.3	**12/15**	0.40 ± 0.71	1.5 ± 3.1	61.3 ± 6.5	**15/15**

a human supervisor during training and execution. Experiments suggest that ThriftyDAgger effectively enables long-horizon robotic manipulation tasks in simulation, on a physical robot, and for a three-robot fleet while limiting burden on a human supervisor. In future work, we hope to apply ideas from ThriftyDAgger to interactive reinforcement learning and larger scale fleets of physical robots. We also hope to study how ThriftyDAgger's performance varies with the target supervisor burden specified via α_H. In practice, α_H could even be time-varying: for instance, α_H may be significantly lower at night, when human operators may have limited availability. Similarly, α_H may be set to a higher value during training than at deployment, when the robot policy is typically higher quality.

Chapter 4

On-Policy Robot Imitation Learning from a Converging Supervisor

In robotics there is significant interest in using human or algorithmic supervisors to train policies via imitation learning [66, 80, 81, 82]. For example, a trained surgeon with experience teleoperating a surgical robot can provide successful demonstrations of surgical maneuvers [83]. Similarly, known dynamics models can be used by standard control techniques, such as model predictive control (MPC), to generate controls to optimize task reward [84, 85]. 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 [85] and model-based control techniques, such as MPC, typically require computationally expensive stochastic optimization techniques to plan over complex dynamics models [86, 87, 88]. 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) [89, 90, 91], 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 [89, 90] or make specific structural assumptions on the supervisor [91].

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 [8, 92, 93, 94]. 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 [86] 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 work 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 [86] 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.

4.1 Related Work

On-policy imitation learning algorithms that directly learn reactive policies from a supervisor were popularized with DAgger [8], 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 [95, 9]. On-policy methods have been applied with both human [96] and algorithmic supervisors [85], 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 [93, 94, 97]. 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 [98, 29]. 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 [89, 90, 91]. 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 [86, 87, 88]. 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. [91] 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 [89, 90], or making specific assumptions on the structure of the model-based controller [91]. 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 [86] as an improving supervisor to achieve fast policy evaluation while maintaining the data efficiency of PETS.

4.2 Converging Supervisor Framework and Preliminaries

4.2.1 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 [8, 93, 94]. 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.

4.2.2 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 4.3.1) and dynamic regret (Section 4.3.2) 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 4.2.1. ***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 4.2.2. ***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 4.2.3. ***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 [99] and Ross, Gordon, and Bagnell [8] 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, Gordon, and Bagnell [8].

Assumption 4.2.1. ***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 4.2.1 preserves strong convexity of the squared loss for an individual sample, which is assumed to be convex in θ.

Assumption 4.2.2. ***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 4.2.3. ***Bounded action space:*** *The action space $\mathcal{A}$ has diameter δ. Equivalently stated: $\delta = \sup_{a_1, a_2 \in \mathcal{A}} \|a_1 - a_2\|$.*

4.3 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 [100], both of which have been utilized in previous on-policy imitation learning analyses [8, 93]. 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, Gordon, and Bagnell [8].

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 [8]. 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 [93, 97]; 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 and Boots [94]. 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.

4.3.1 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 [99] and DAgger [8]; 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 4.3.1.

Definition 4.3.1. *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.*

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

However, we instead analyze the more difficult regret metric presented in Definition 4.3.2 below.

Definition 4.3.2. *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.*

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

Theorem 4.3.1. $\text{Regret}_N^S(\psi_N)$ *can be bounded above as follows:*

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

Theorem 4.3.1 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 [99] and DAgger [8] 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. Corollary 4.3.1 identifies when this is the case.

Corollary 4.3.1. *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:*

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

4.3.2 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 [94, 96]. Recent analyses have turned to dynamic regret [93, 94], 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 [93]. 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 [93, 94, 97]. 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 [97]. 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 Definition 4.3.3 below.

Definition 4.3.3. *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*.*

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

However, similar to the static regret analysis in Section 4.3.1, we seek to analyze the dynamic regret with respect to labels from the last observed supervisor ψ_N, which is defined as follows.

Definition 4.3.4. *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*.*

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

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

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

Given the notion of supervisor convergence discussed in Corollary 4.3.1, Corollary 4.3.2 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 4.3.2. *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 [93]. However, as in [93, 94], 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 4.3.1. *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 [101, 102], and Lemma 4.3.2 shows that with it, sublinear $\text{Regret}_N^D((\psi_i)_{i=1}^N)$ can be achieved using results from predictable online learning [97].

Lemma 4.3.2. *If Assumption 4.3.1 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 Corollary 4.3.2 and Lemma 4.3.2 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 4.3.2. *If* $\forall s \in \mathcal{S},\ \forall N > i, \|\psi_i(s) - \psi_N(s)\| \leq f_i$ *where* $\lim_{i\to\infty} f_i = 0$ *and under the assumptions in Lemma 4.3.2, 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* $\text{Regret}_N^D(\psi_N)$*.*

4.4 Converging Supervisors for Deep Continuous Control

Sun et al. [91] 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 [89, 90]. 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 [86]. We collect policy rollouts from a model-free learner policy and refit the policy on each episode using DAgger [8] 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.

4.5 Experiments

The method presented in Section 4.4 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. [91] 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 [103].
2. **Twin Delayed Deep Deterministic policy gradients (TD3)**: State-of-the-art model-free RL algorithm [62] which uses target networks and delayed policy updates to improve DDPG [104], 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 [102].

4.5.1 Simulation Experiments

We consider the PR2 Reacher and Pusher continuous control MuJoCo domains from Chua et al. [86] (Figure 4.1) 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 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 4.1, 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.

4.5.2 Physical Robot Experiments

We also test CSF with a neural network policy on a physical da Vinci Surgical Robot (dVRK) [105] 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

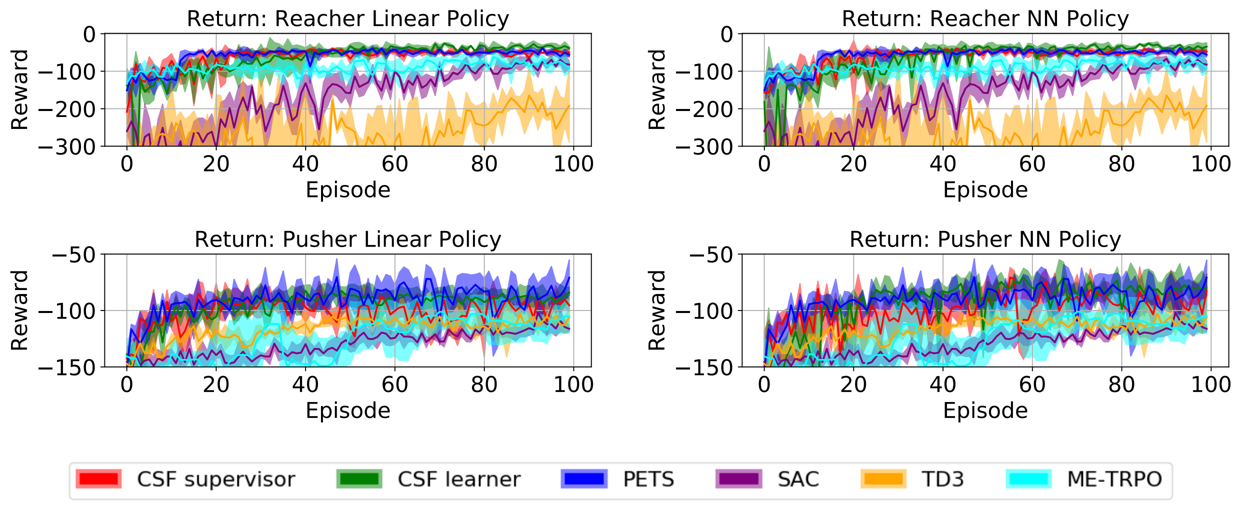


Figure 4.1: **Simulation experiments:** Training curves for the CSF learner, CSF supervisor, PETS, and baselines for the MuJoCo Reacher (top) and Pusher (bottom) tasks for a linear (left) and neural network (NN) policy (right). The linear policy is trained via ridge-regression with regularization parameter $\alpha = 1$, satisfying the strongly-convex loss assumption in Section 4.2. 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.

find that the CSF learner is able to track the PETS supervisor effectively (Figure 4.2) and provide up to a 22x speedup in policy query time (Table 4.1). 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.

4.6 Discussion and Future Work

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 accel-

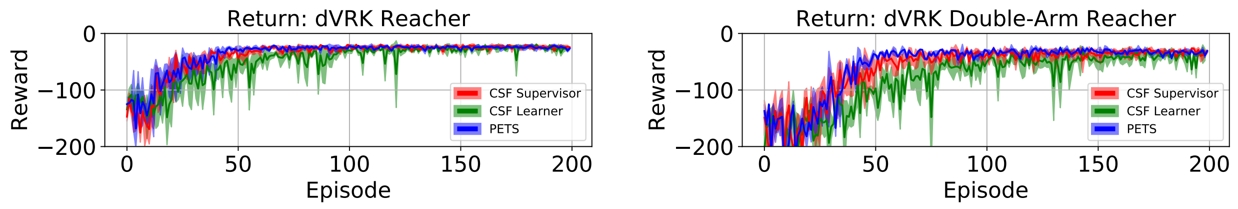


Figure 4.2: **Physical experiments:** Training curves for the CSF learner, CSF supervisor and PETS on the da Vinci surgical robot with a neural network policy. The CSF learner is able to track the CSF supervisor and PETS effectively and can be queried up to 20x faster than PETS. However, due to control 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 4.1: **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	$(0.78 \pm 0.02, 8.43 \pm 1.07)$	$(0.88 \pm 0.07, 12.97 \pm 0.77)$

erating 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

Reinforcement Learning from Suboptimal Demonstrations

Chapter 5

ABC-LMPC: Safe Learning MPC for Stochastic Nonlinear Dynamical Systems with Adjustable Boundary Conditions

Model predictive control (MPC) has seen significant success in a variety of robotic tasks [106, 86, 107], and there is substantial experimental and theoretical work demonstrating that the resulting closed loop system performs well on challenging tasks in stochastic dynamical systems [106, 108, 109, 110]. In this work, we build on the recently proposed learning model predictive control (LMPC) framework [111, 110, 109]. 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 [111, 110, 109] presents a novel class of reference-free control strategies which utilize MPC to iteratively improve upon a suboptimal controller 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 [110, 109] and deterministic nonlinear systems [111] if the MPC problem can be solved exactly. However, while LMPC-based control strategies exhibit a variety of desirable theoretical properties [111, 110, 109] and have been shown to work well on practical problems on physical robotic systems [112, 106], 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 [109] 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. The contributions of this work are (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 controller can adapt to novel start states and goal sets while consistently and efficiently completing tasks during learning.

5.1 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 [106, 86, 107] and controls communities [113, 114, 115, 116, 117, 118]. Prior work in learning for model predictive control has focused on estimating the following three components used to design MPC policies: *i*) a model of the system [114, 116, 119, 112, 117, 86, 107, 106], *ii*) a safe set of states from which the control task can be completed using a known safe policy [120, 121, 122, 123] and *iii*) a value function [106, 109, 110, 124], which for a given safe policy, maps each state of the safe set to the closed-loop cost to complete the task. The most closely related works, both by Rosolia et. al. [109, 110], 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 controller improvement if the MPC problem can be solved exactly. Furthermore, Thananjeyan* et al. [106] propose a practical reinforcement learning algorithm using these strategies to learn policies for nonlinear systems. However, [110, 106] are limited to the iterative learning control setting, and although [109] presents a strategy for controller domain expansion, the method is limited to linear systems and requires the user to precisely specify an expansion direction. In this work, we build on this framework by (1) extending the theoretical results to prove that under similar assumptions, LMPC based controllers 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 controller 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 [125, 126, 127] 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. [125] and Resnick et al. [126] 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. [128]. However, [128] 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 controller feasibility and convergence guarantees. There is also recent interest in goal-conditioned RL [129, 130]. The most relevant prior work in this area is hindsight experience replay [131], 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 [132, 133]. 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 [134].

5.2 Problem Statement

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

$$x_{t+1} = f(x_t, u_t, w_t) \tag{5.1}$$

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 [109], 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 $C(\cdot,\cdot)$:

$$J^{\pi^j}(x_0^j) = \mathop{\mathbb{E}}_{w_0^j}\left[C(x_0^j, \pi^j(x_0^j)) + J^{\pi^j}(f(x_0^j, u_0^j, w_0^j))\right] \tag{5.2}$$

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:

$$\begin{aligned} J^{j,*}_{0\to\infty}(x^j_0) = \min_{\pi^j(.)} \quad & J^{\pi^j}(x^j_0) \\ \text{s.t.} \quad & x^j_{t+1} = f(x^j_t, u^j_t, w^j_t) \\ & u^j_t = \pi^j(x^j_t) \\ & x^j_t \in \mathcal{X}, u^j_t \in \mathcal{U} \\ & \forall w^j_t \in \mathcal{W}, t \in \{0, 1, \ldots\} \end{aligned} \tag{5.3}$$

In this chapter, we present a strategy to iteratively design a feedback policy $\pi^j(.) : \mathcal{F}^j_\mathcal{G} \subseteq \mathcal{X} \to \mathcal{U}$, where $\mathcal{F}^j_\mathcal{G}$ 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 controller 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 $x^j_0 = x^l_0$ where $j < l$, expected trajectory cost is non-increasing ($J^{\pi^j}(x^j_0) \geq J^{\pi^{j+1}}(x^{j+1}_0)$), and (iv) exploration: the domain of the control policy does not shrink over iterations ($\mathcal{F}^j_\mathcal{G} \subseteq \mathcal{F}^{j+1}_\mathcal{G}$ for all goal sets $\mathcal{G}$ sampled up to iteration j). In Section 5.3.3, 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 5.2.1. ***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 5.2.1. ***Robust Control Invariant Set:*** *As in Rosolia and Borrelli [109], we define a robust control invariant set $\mathcal{A} \subseteq \mathcal{X}$ with respect to dynamics $f(x, u, w)$ and policy class Π as a set where $\forall x \in \mathcal{A}$, $\exists \pi \in \Pi$ s.t. $f(x, \pi(x), w) \in \mathcal{A}, \pi(x) \in \mathcal{U}$, $\forall w \in \mathcal{W}$.*

Assumption 5.2.2. ***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.*

5.3 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.

5.3.1 Safe Set

We first recall the definition of a robust reachable set as in [109]:

Definition 5.3.1. ***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* (5.1) *in closed loop with π at iteration j:*

$$\mathcal{R}_{t+1}^{\pi}(x_0^j) = \big\{x_{t+1} \in \mathbb{R}^n \mid \exists w_t \in \mathcal{W},\ x_t \in \mathcal{R}_t^{\pi}(x_0^j),\ x_{t+1} = f(x_t, \pi(x_t), w_t)\big\} \tag{5.4}$$

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 [109].

Definition 5.3.2. ***Safe Set:*** *The safe set $\mathcal{SS}_{\mathcal{G}}^j$ contains the full evolution of the system at iteration j,*

$$\mathcal{SS}_{\mathcal{G}}^j = \left\{\bigcup_{t=0}^{\infty} \mathcal{R}_t^{\pi^j}(x_0^j) \bigcup \mathcal{G}\right\}. \tag{5.5}$$

Note that (5.5) is robust control invariant by construction [109]. We could set $\mathcal{SS}_{\mathcal{G}}^0 = \mathcal{G}$ or initialize the algorithm with a nominal controller π^0. This enables the algorithm to naturally incorporate demonstrations to speed up training.

Definition 5.3.3. ***Expected Cost:*** *The expected cost of π^j from start state x_0^j is defined as*

$$J^{\pi^j}(x_0^j) = \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] \tag{5.6}$$

5.3.2 Value Function

Definition 5.3.4. ***Value Function:*** *Recursively define the value function of π^j in closed-loop with* (5.3) *as:*

$$L_{\mathcal{G}}^{\pi^j}(x) = \begin{cases} \mathop{\mathbb{E}}_{w}\left[C(x, \pi^j(x)) + L_{\mathcal{G}}^{\pi^j}(f(x, \pi^j(x), w))\right] & x \in \mathcal{SS}_{\mathcal{G}}^j \\ +\infty & x \notin \mathcal{SS}_{\mathcal{G}}^j \end{cases} \tag{5.7}$$

Let $V_{\mathcal{G}}^{\pi^j}(x) = \min_{k \in \{0, \dots j\}} L_{\mathcal{G}}^{\pi^k}(x)$, which is the expected cost-to-go of the best performing prior controller at x.

Observe that $L_{\mathcal{G}}^{\pi^j}$ is defined only on $\mathcal{SS}_{\mathcal{G}}^j$, and $J^{\pi^j} = L_{\mathcal{G}}^{\pi^j}$ on $\mathcal{SS}_{\mathcal{G}}^j$. In the event a nominal controller π^0 is used, we require the following assumption on the initial safe set $\mathcal{SS}_{\mathcal{G}}^0$, which is implicitly a restriction on π^0 for start state x_0^0.

Assumption 5.3.1. ***Safe Set Initial Condition:*** *If a nominal controller π^0 is used, then $\forall x \in \mathcal{SS}_{\mathcal{G}}^0,\ L_{\mathcal{G}}^{\pi^0}(x) < \infty$.*

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

5.3.3 Transfer to Novel Goal Sets

While Rosolia and Borrelli [109] studies tasks with fixed goal sets, here we show how the safe set and value function can be modified to transfer the learned controller 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 5.3.5. ***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:*

$$\mathcal{SS}^j_{\mathcal{H}} = \begin{cases} \bigcup_{k=0}^{k^*} \mathcal{R}_k^{\pi^j} \bigcup \mathcal{H} & \max\limits_{k\in\mathbb{N}} \mathbb{1}\{\mathcal{R}_k^{\pi^j}(x_0^j) \subseteq \mathcal{H}\} = 1 \\ \mathcal{H} & \textit{otherwise} \end{cases} \tag{5.8}$$

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

We also redefine the value function as follows:

Definition 5.3.6. ***Goal Conditioned Value Function:*** *Recursively define the goal-conditioned value function of π^j in closed-loop with* (5.3) *as:*

$$L^{\pi^j}_{\mathcal{H}}(x) = \begin{cases} \mathbb{E}_w \left[C(x, \pi^j(x)) + L^{\pi^j}(f(x, \pi^j(x), w)) \right] & x \in \mathcal{SS}^j_{\mathcal{H}} \setminus \mathcal{H} \\ 0 & x \in \mathcal{H} \\ +\infty & x \notin \mathcal{SS}^j_{\mathcal{H}} \end{cases} \tag{5.9}$$

Define $V^{\pi^j}_{\mathcal{H}}(x) = \min\limits_{k\in\{0,\ldots j\}} L^{\pi^k}_{\mathcal{H}}(x)$ *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.

5.4 Controller Design

Here we describe the controller design for optimizing the task cost function while satisfying state and input constraints (Section 5.4.1), and discuss how this can be extended to iteratively expand the controller domain (Section 5.4.2). 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 5.3.5 and 5.3.6. See Figure 5.1 for an illustration of the full ABC-LMPC controller optimization procedure.

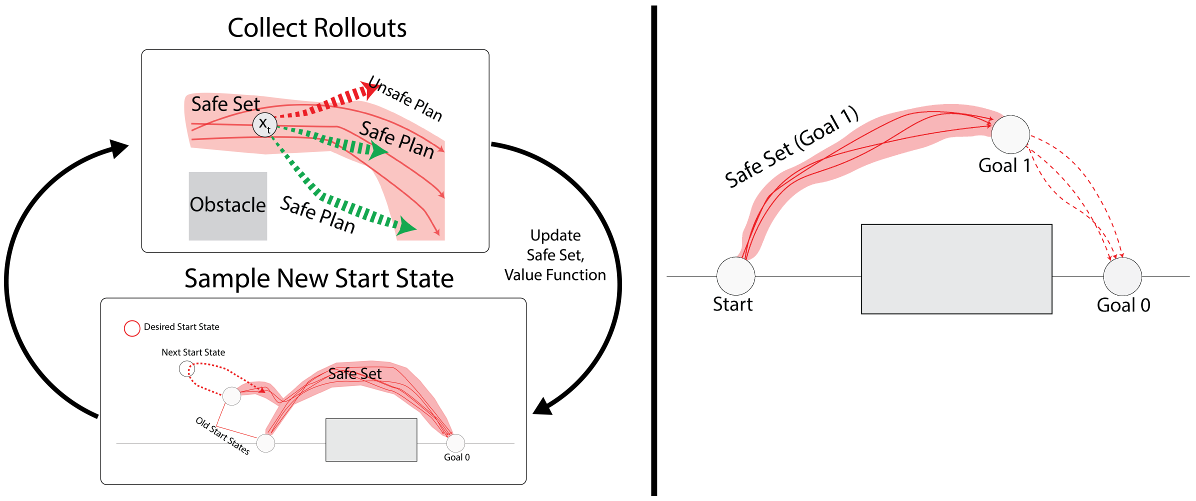


Figure 5.1: **ABC-LMPC Iterative Algorithm (Left):** ABC-LMPC alternates between (1) collecting rollouts under the current policy π^j given $\mathcal{SS}^j_{\mathcal{G}_0}$ and $L^{\pi^j}_{\mathcal{G}_0}$ (by optimizing (5.10)), (2) updating $\mathcal{SS}^{j+1}_{\mathcal{G}_0}$ and $L^{\pi^{j+1}}_{\mathcal{G}_0}$ given the new rollouts, and (3) expanding the controller domain towards a desired start state (by optimizing (5.13)); **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 ($L^{\pi^j}_{\mathcal{G}_1}$).

5.4.1 Task Driven Optimization

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

$$\begin{aligned}
J^j_{t\to t+H}(x^j_t) = \min_{\pi_{t:t+H-1|t}\in\Pi^H} \quad & \mathop{\mathbb{E}}_{w^j_{t:t+H-1}}\left[\sum_{i=0}^{H-1} C(x^j_{t+i|t}, \pi_{t+i|t}(x^j_{t+i|t})) + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H|t})\right] \\
\text{s.t.} \quad & x^j_{t+i+1|t} = f(x^j_{t+i|t}, \pi_{t+i|t}(x^j_{t+i}), w_{t+i})\ \forall i \in \{0,\dots,H-1\} \\
& x^j_{t+H|t} \in \bigcup_{k=0}^{j-1} \mathcal{SS}^k_{\mathcal{G}},\ \forall w^j_{t:t+H-1} \in \mathcal{W}^H \\
& x^j_{t:t+H|t} \in \mathcal{X}^{H+1},\ \forall w^j_{t:t+H-1} \in \mathcal{W}^H
\end{aligned} \tag{5.10}$$

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 (5.10) be $\pi^{*,j}_{t:t+H-1|t}$. Then, execute the first policy at x^j_t:

$$u^j_t = \pi^j(x^j_t) = \pi^{*,j}_{t|t}(x^j_t) \tag{5.11}$$

Solving 5.10 is typically intractable in practice, so we discuss practical approximations we make to the designed algorithm in Section 5.6.

5.4.2 Start State Expansion

We now describe the control strategy for expanding the controller 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 controller 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 controller domain as follows:

$$\mathcal{F}_{\mathcal{G}}^j = \{x \mid \exists \pi_{0:H-1} \in \Pi^H \text{ s.t. } \mathcal{R}_H^{\pi_{0:H-1}}(x) \subseteq \bigcup_{k=0}^{j-1} \mathcal{SS}_{\mathcal{G}}^k\} \tag{5.12}$$

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 controller 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 5.5), 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 controller terminates in some state $x_{H'}^j \in \bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k$. In Section 5.6, 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 controller feasibility. The sequence of controllers can computed by solving the following 1-step trajectory optimization problem:

$$\begin{aligned}
\pi_{E,0:H'-1}^j = \underset{\pi_{0:H'-1} \in \Pi^{H'}}{\operatorname{argmin}} \quad & \underset{w_{0:H'-2}^j}{\mathbb{E}} \left[\sum_{i=0}^{H'-1} C_E^j(x_i^j, \pi_i(x_i^j)) \right] \\
\text{s.t.} \quad & x_{i+1}^j = f(x_i^j, \pi_i(x_i^j), w_i),\ \forall i \in \{0, \dots, H'-1\} \\
& x_{H'}^j \in \bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^k,\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \\
& x_{0:H'}^j \in \mathcal{X}^{H'+1},\ \forall w_{0:H'-2} \in \mathcal{W}^{H'-1}
\end{aligned} \tag{5.13}$$

Let $\mathcal{M} = \left(\mathcal{R}_i^{\pi_{E,0:H'-1}^j}(x_S^j) \right)_{i=0}^{H'}$, the set of all states reachable in H' steps by π_E and let $\mathcal{M}_H = \left(\mathcal{R}_i^{\pi_{E,0:H'-1}^j}(x_S^j) \right)_{i=\max(H'-H,\ 0)}^{H'}$. Note that $\forall x \in \mathcal{M}_H$, the controller initialized at x

can be robustly guided to $\bigcup_{k=0}^{j} \mathcal{SS}_{\mathcal{G}}^{k}$ 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_E^j could be executed instead to generate these future start states. We could also use the exploration policy to explicitly augment the value function $L_{\mathcal{G}}^{\pi^j}$ and safe set $\mathcal{SS}_{\mathcal{G}}^j$ and thus $\mathcal{F}_{\mathcal{G}}^j$. This could be used for general domain expansion instead of directed expansion towards a desired start state.

5.5 Properties of ABC-LMPC

In this section, we study the properties of the controller constructed in Section 5.4. 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 $\mathcal{H}$. See Appendix D.1 for all proofs.

Lemma 5.5.1. ***Recursive Feasibility:*** *Consider the closed-loop system* (5.10) *and* (5.11)*. Let the safe set $\mathcal{SS}_{\mathcal{G}}^j$ be defined as in* (5.5)*. If assumptions 5.2.1-5.3.1 hold and $x_0^j \in \mathcal{F}_{\mathcal{G}}^j$, then the controller induced by optimizing* (5.10) *and* (5.11) *is feasible almost surely for* $t \geq 0$ *and* $j \geq 0$*. Equivalently stated,* $\underset{w_{0:H-1}^j}{\mathbb{E}} [J_{t\to t+H}^j(x_t^j)] < \infty$*,* $\forall t, j \geq 0$*.*

Lemma 5.5.1 shows that the controller 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 controller is guaranteed to be finite. The following lemma establishes convergence in probability to the goal set given initialization within the controller domain.

Lemma 5.5.2. ***Convergence in Probability:*** *Consider the closed-loop system defined by* (5.10) *and* (5.11)*. Let the sampled safe set $\mathcal{SS}_{\mathcal{G}}^j$ be defined as in* (5.5)*. Let assumptions 5.2.1-5.3.1 hold and $x_0^j \in \mathcal{F}_{\mathcal{G}}^j$. 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:* $\lim_{t\to\infty} P(x_t^j \notin \mathcal{G}) = 0$*.*

Theorem 5.5.1. ***Iterative Improvement:*** *Consider system* (5.1) *in closed-loop with* (5.10) *and* (5.11)*. Let the sampled safe set $\mathcal{SS}^j$ be defined as in* (5.5)*. Let assumptions 5.2.1-5.3.1 hold, then the expected cost-to-go* (5.6) *associated with the control policy* (5.11) *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.*

Theorem 5.5.1 extends prior results [109], 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 5.2.1. The following result implies that the controller domain is non-decreasing.

Lemma 5.5.3. ***Controller domain expansion:*** *The domain of* π^j *is an non-decreasing sequence of sets:* $\mathcal{F}^j_\mathcal{G} \subseteq \mathcal{F}^{j+1}_\mathcal{G}$.

5.6 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 (5.10) and exploration (5.13) 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 controller design is implemented and how optimization is performed. See Appendix D.2 for further implementation details.

5.6.1 Sample-Based Safe Set

In practice, as in [109], we approximate the safe set $\mathcal{SS}^j_\mathcal{G}$ using samples from the closed loop system defined by (5.10) and (5.11). To do this, we collect R closed-loop trajectories at iteration j, each of length T as in [109] 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 [109]: $\mathbf{x}^j(\mathbf{w}^j_i) = \left[x^j_0(\mathbf{w}^j_i), \dots, x^j_T(\mathbf{w}^j_i)\right]$. As in [109], 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 [109]. To obtain a continuous approximation of the safe set for planning, we use the same technique as [106], 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 $\rho^\mathcal{G}_\alpha$.

5.6.2 Start State Expansion Strategy

To provide a sample-based approximation to the procedure from Section 5.4.2, we sample states x_S^j 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_S^j 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 5.4.2). 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 controller 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 controller to desired specifications while maintain controller 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 5.7.3). 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.

5.6.3 Goal Set Transfer

We practically implement the goal set transfer strategy in Section 5.3.3 by fitting a new density model $\rho^{\mathcal{H}}_\alpha$ 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 controller can seamlessly transfer to $\mathcal{H}$. If this is not the case, the controller domain for $\mathcal{H}$ must be expanded from $\mathcal{H}$ until it intersects many trajectories in the original domain.

5.6.4 ABC-LMPC Optimization Procedure

As in prior work on MPC for nonlinear control [86, 106], we solve the MPC optimization problem in (5.10) over sampled open loop sequences of controls using the cross entropy method (CEM) [135]. In practice, we implement the terminal safe set constraints and state-space constraints in (5.10) and (5.13) by imposing a large cost on sampled action sequences which violate constraints when performing CEM. We use a probabilistic ensemble of 5 neural networks to approximate $L^{\pi^j}_{\mathcal{G}}(x)$ as in [106]. In contrast to [106], 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 [86].

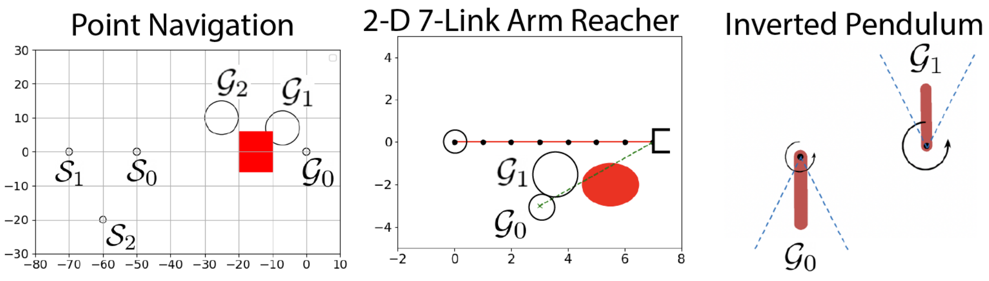


Figure 5.2: **Experimental Domains:** We evaluate ABC-LMPC on three stochastic domains: a navigation domain with an obstacle, a 2D 7-link arm reacher domain with an obstacle, and an inverted pendulum domain. In the first two domains, suboptimal demonstrations are provided, while no demonstrations are provided for the inverted pendulum task.

5.7 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 5.7.1 we describe the experimental domains, in Section 5.7.2, we evaluate the controller with fixed start states and goal sets, in Section 5.7.3, we expand the controller domain iteratively toward a desired start state far from the goal set, in Section 5.7.4, we switch the goal set during learning, and finally in Section 5.7.5 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 [106]. 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 [106] after 10 iterations of training to evaluate the quality of the learned controller. For all experiments, we use $R = 5$ closed-loop trajectories from the current controller to estimate $\tilde{\mathcal{SS}}^j_{\mathcal{G}}$ and perform start state expansion. Experimental domains have comparable stochasticity to those in [109]. See Appendix D.3 for further details about experimental, optimization, and environment parameters.

5.7.1 Experimental Domains

Point Mass Navigation: We consider a 4-dimensional (x, y, v_x, v_y) navigation task as in [106], 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.

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 [136]. 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 include a circular obstacle of radius 1 in the environment which the kinematic chain must avoid.

Inverted Pendulum: This environment is a noisy inverted pendulum task adapted from OpenAI Gym [137]. We introduce truncated Gaussian process noise in the dynamics.

5.7.2 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 optimization parameters and other experimental details, see Appendix D.3. The controller rapidly and significantly improves upon demonstrations for both domains (Figure 5.3). The controller achieves comparable cost to SAVED for both tasks and never violates constraints during learning.

5.7.3 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 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.

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 controller 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 controller reaches the desired start state while consistently maintaining feasibility during learning (Table 5.1).

We then consider a similar Reacher task using the same suboptimal demonstrations from Section 5.7.2. The desired start end effector position is $(-1, 0)$, and $C_E(x, u)$ measures the

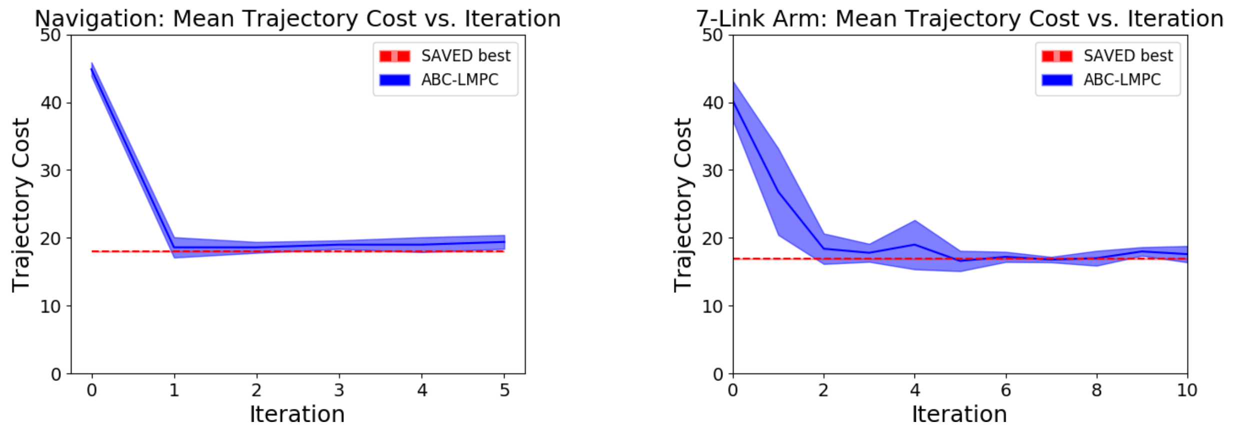


Figure 5.3: **Fixed Start, Single Goal Set Experiments:** Learning curves for ABC-LMPC averaged over $R = 5$ rollouts per iteration on simulated continuous control domains when the start state and goal set is held fixed during learning. Performance of the demonstrations is shown at iteration 0, and the controller performance is shown thereafter. **Point Mass Navigation:** The controller 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 controller 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 controller quickly converges to the best cost produced by SAVED.

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 controller is able to start at the desired start state while maintaining feasibility during learning (Table 5.1). On both domains, the controller achieves comparable performance to SAVED when trained with demonstrations from that start state and the controller successfully expands its domain while rapidly achieving good performance at the new states. Constraints are never violated during learning for all experiments.

5.7.4 Goal Set Transfer

ABC-LMPC is trained as in Section 5.7.2, but after a few iterations, the goal set is changed to a new goal set that is in the controller 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 5.4 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

Table 5.1: **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 controller feasibility throughout learning. Furthermore, the controller 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 controller 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 $\pm$ 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

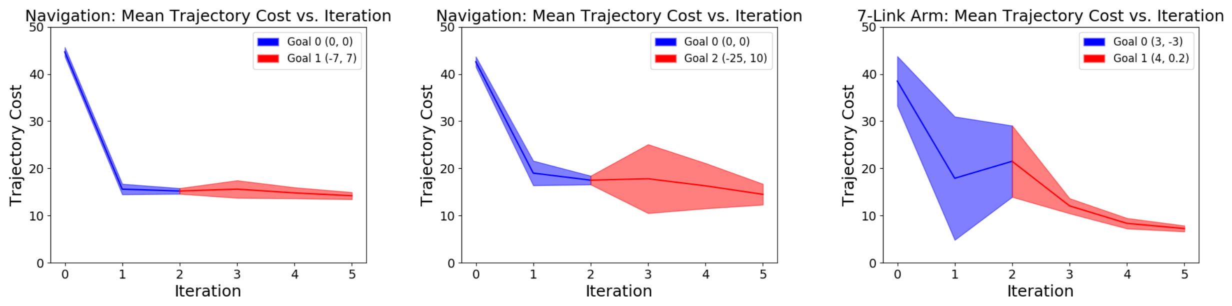
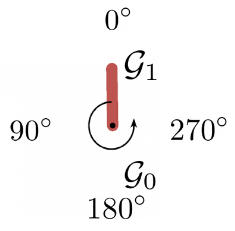


Figure 5.4: **Goal Set Transfer Learning:** In this experiment, the goal set is switched to to a new goal set at iteration 3 and we show a learning curve which indicates performance on both the first goal (blue) and new goal (red). The controller is re-trained as in Section 5.6.3 to stabilize to the new goal. The controller immediately is able to perform the new task and never hits the obstacle. Results are plotted over $R = 5$ controller rollouts per iteration.

centered at $\mathcal{G}_1 = (4, 0.2)$ with radius 1 after 2 iterations of training. Results are shown in Figure 5.4. In both domains, ABC-LMPC seamlessly transfers to the new goal by leveraging prior experience to train a new set of value functions.

Table 5.2: **Pendulum Swing Up Experiment:** We iteratively expand the controller domain outward from a goal set centered around the downward orientation ($\mathcal{G}_0$) towards the upward orientation until the controller 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 controller maintains feasibility throughout and seamlessly transitions to $\mathcal{G}_1$. The trajectory costs are presented as: mean $\pm$ 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



5.7.5 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 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 controller 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 controller 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 5.2). $C_E(x, u)$ measures the distance between the pendulum's orientation and the desired start state's orientation.

5.8 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 controller 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 controllers when dynamics are learned from experience.

Chapter 6

SAVED: Safe Deep Model-Based RL for Sparse Cost Robotic Tasks

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 [87, 86, 112, 138, 139, 108, 140]. However, past work in deep MBRL typically requires dense hand-engineered cost functions, which are hard to design and can lead to unintended behavior [141]. 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 environments 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 [143] 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.

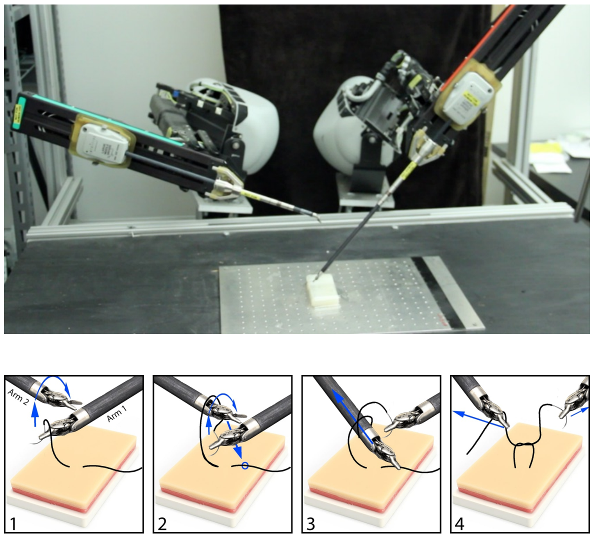


Figure 6.1: SAVED is able to safely learn maneuvers on the da Vinci surgical robot, which is difficult to precisely control [142]. We demonstrate that SAVED is able to optimize inefficient human demonstrations of a surgical knot-tying task, substantially improving on demonstration performance with just 15 training iterations.

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 work 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.

6.1 Related work

There is significant interest in model-based planning and deep MBRL [138, 139, 140, 144, 86, 87] due to the improvements in sample efficiency when planning over learned dynamics compared to model-free methods for continuous control [103, 62]. 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 [145] and DDPG [146, 147], but model-free methods are typically less sample efficient and cannot anticipate constraint violations since they learn reactive policies [148]. Demonstrations have also been leveraged in model-based algorithms, such as in motion planning with known dynamics [149] and for seeding a learned dynamics model for fast online adaptation using iLQR and a dense cost [140], 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. [29] 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 [150]. Rosolia, Zhang, and Borrelli [151] and Rosolia and Borrelli [109, 111] 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, Zhang, and Borrelli [151] and Rosolia and Borrelli [109, 111], 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 [152], typically focusing on exploration while satisfying a set of explicit constraints [153, 154, 155], satisfying specific stability criteria [156], or formulating planning via a risk sensitive Markov Decision Process [157, 158]. 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.

6.2 Safety Augmented Value Estimation from Demonstrations (SAVED)

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.

6.2.1 Assumptions and Preliminaries

In this work, 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 $\mathbb{1}(x \in \mathcal{X})$, 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 [147].

Finally, we define the notion of a safe set to enable constrained policy improvement, which is described further in Section 6.2.3. Recent MPC literature [111] 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 [151, 109, 111]. For a trajectory at iteration k, given by $x^k = \{x_t^k | t \in \mathbb{N}\}$, we define the *sampled safe set* as

$$\mathcal{SS}^j = \bigcup_{k \in \mathcal{M}^j} x^k \tag{6.1}$$

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 and Borrelli [111]. $\mathcal{SS}^j$ 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

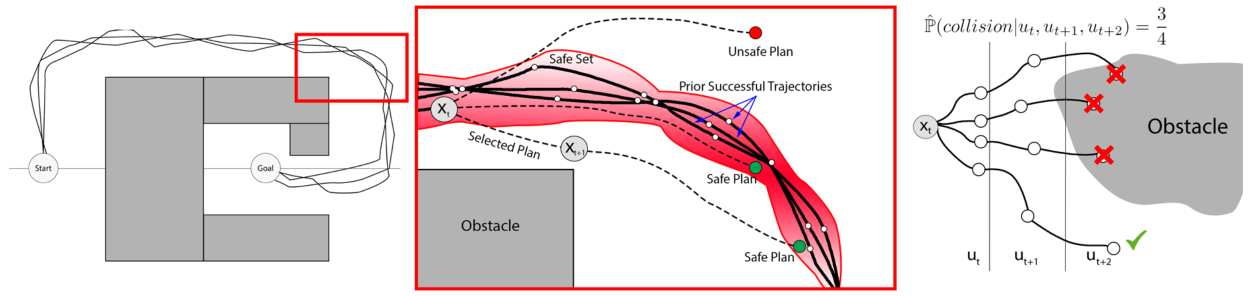


Figure 6.2: **Task Completion Driven Exploration (left):** A density model is used to represent the region in state space where the agent has high confidence in task completion; trajectory samples over the learned dynamics that do not have sufficient density at the end of the planning horizon are discarded. 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.

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.

6.2.2 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 [159]. Additionally, the planner receives no feedback or information about task progress when using the indicator task functions used in this work. Thus, to guide exploration in temporally-extended tasks, we solve the problem in equation 6.2a, which includes a learned value function in the 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^{π}_{ϕ}, which provides a terminal cost estimate for the current policy at the end of the planning horizon.

While prior work in deep MBRL [87, 86] 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^π_ϕ are each represented with a finite probabilistic ensemble of n neural networks (in this work we pick $n = 5$), as is used to represent system dynamics in Chua et al. [86]. 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 [86]. 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 6.2c 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 6.2.3, we discuss the methods used for task completion driven exploration and in Section 6.2.4, 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:

$$u^*_{t:t+H-1} = \underset{u_{t:t+H-1}\in\mathcal{U}^H}{\operatorname{argmin}} \mathbb{E}_{x_{t:t+H}}\left[\sum_{i=0}^{H-1} C(x_{t+i}, u_{t+i}) + V^\pi_\phi(x_{t+H})\right] \tag{6.2a}$$

$$\text{s.t. } x_{t+i+1} \sim f_\theta(x_{t+i}, u_{t+i}) \; \forall i \in \{0, \dots, H-1\} \tag{6.2b}$$

$$\rho_\alpha(x_{t+H}) > \delta, \mathbb{P}\left(x_{t:t+H} \in \mathcal{X}^{H+1}\right) \geq \beta \tag{6.2c}$$

6.2.3 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

Algorithm 2 Safety Augmented Value Estimation from Demonstrations (SAVED)

```
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 $i \in \{1, \dots, N\}$ do
    Sample $x_0$ from start state distribution
    for $t \in \{1, \dots, T-1\}$ do
      Pick $u^*_{t:t+H-1}$ by solving eq. 6.2 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
```

nonlinear dynamics [111, 151, 109]. The way we constrain exploration in SAVED builds on this prior work, but we note that unlike Rosolia and Borrelli [111], Rosolia, Zhang, and Borrelli [151], and Rosolia and Borrelli [109], SAVED is designed for settings in which dynamics are completely unknown, nonlinear, and stochastic. As illustrated in Figure 6.2, 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 6.2c). Since the tasks considered in this work 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 6.2.4, with further details in the supplementary material. In future work, we will investigate implicit density estimation techniques to scale to high-dimensional settings.

6.2.4 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) [86]. Each control sequence is simulated multiple times over the stochastic dynamics model as in [86] and the average return of the simulations is used to score the sequence. However, unlike Chua et al. [86], we implement chance constraints by discarding control sequences if more than $100 \cdot (1 - \beta)\%$ of the simulations violate constraints (constraint 6.2c), 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 6.2. The task completion constraint (Section 6.2.3) 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.

6.2.5 Algorithm Pseudocode

We summarize SAVED in Algorithm 2. 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 6.2 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.

6.3 Experiments

We evaluate SAVED on simulated continuous control benchmarks and on real robotic tasks with the da Vinci Research Kit (dVRK) [105] 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 6.2.1), 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. [86]. To plan over the dynamics, the TS-∞ trajectory sampling method from [86] 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.

6.3.1 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 6.2.4. 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 [86] 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 [103], where demo transitions are used for training initially.

5. **Overcoming Exploration in Reinforcement Learning from Demonstrations (OEFD)**: Model-free algorithm from Nair et al. [147] 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 6.2.3.

6.3.2 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 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 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 $\beta = 1$.

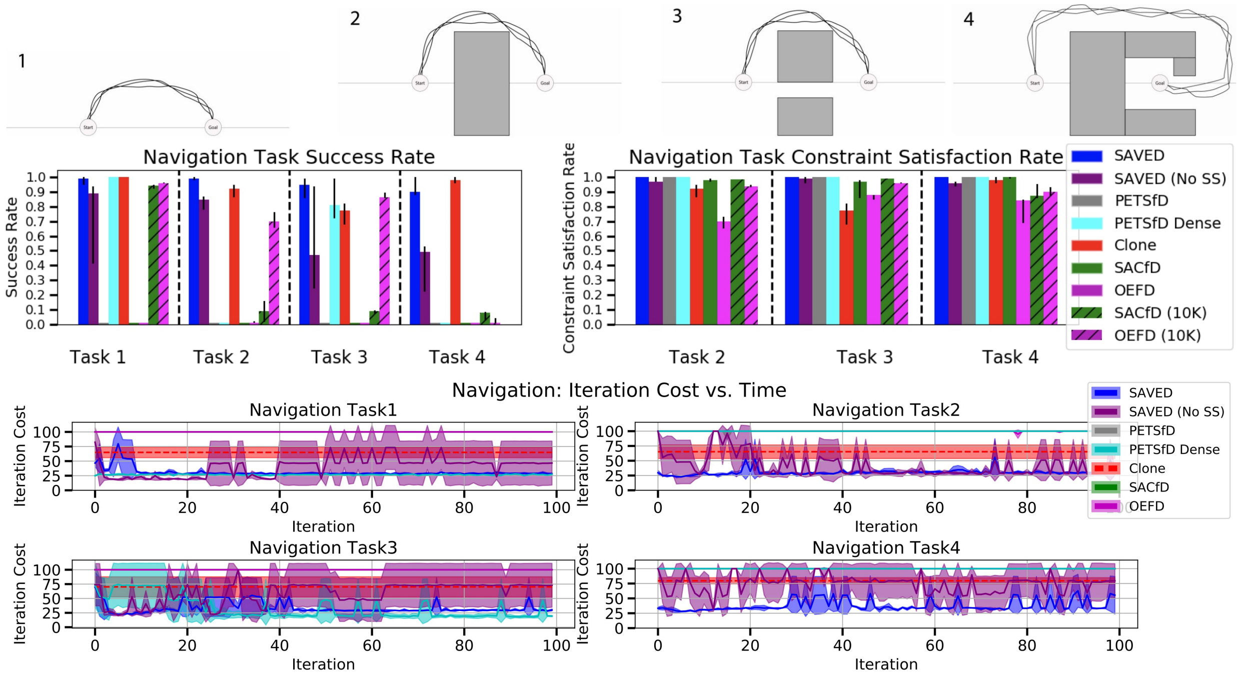


Figure 6.3: **Navigation Domains:** SAVED is evaluated on 4 navigation tasks. Tasks 2-4 contain obstacles, and task 3 contains a channel for passage to $\mathcal{G}$ near the x-axis. SAVED learns significantly faster than all RL baselines on tasks 2 and 4. In tasks 1 and 3, SAVED has lower iteration 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.

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 6.3) 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

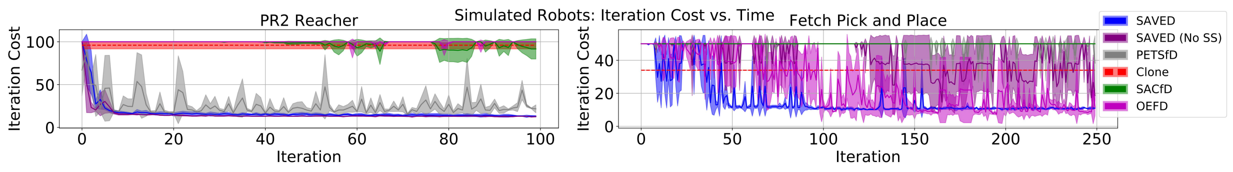


Figure 6.4: **Simulated Robot Experiments Performance:** SAVED achieves better performance than all baselines on both tasks. We use 20 demonstrations with average iteration cost of 94.6 for the reacher task and 100 demonstrations with average iteration cost of 34.4 for the pick and place task. For the 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.

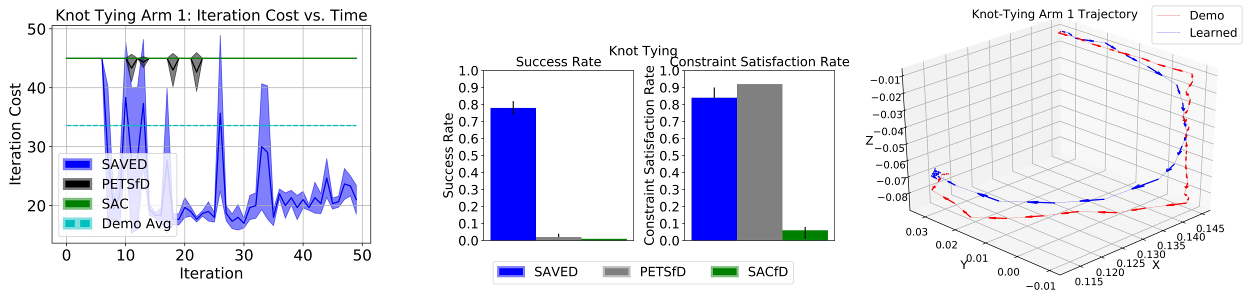


Figure 6.5: **Physical Surgical Knot-Tying: Training Performance:** After just 15 iterations, the agent completes the task relatively consistently with only a few failures, and converges to a iteration cost of 22, faster than demos, which have an average iteration cost of 34. In the first 50 iterations, 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.

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.

6.3.3 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. [86] with a fixed goal and on a pick and place task with a simulated, position-controlled Fetch robot from [160]. 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 6.4) 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.

6.3.4 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) [105]. The dVRK is cable-driven and has relatively imprecise controls, motivating model learning [142]. 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 [161]. 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 6.1. 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 6.5) 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.

6.4 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 7

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

Visual planning over learned forward dynamics models is a popular area of research in robotic control from images [162, 163, 164, 165, 166, 167, 168], 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 [169], especially for long-horizon tasks. Second, while many prior reinforcement learning methods have been successfully applied to image-based control tasks [170, 171, 172, 173, 174], 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 [111, 175], 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 [111, 176, 106], 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 [111, 176, 109], using them for controller design for practical problems requires developing continuous approximations at the expense of maintaining theoretical guarantees [106]. 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 [111] or via kernel density estimation with a tophat kernel function [106]. 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 [111, 176, 177].

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

7.1 Related Work

7.1.1 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 [150]. Rosolia, Zhang, and Borrelli [151] and Rosolia and Borrelli [109, 111] 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, Zhang, and Borrelli [151] and Rosolia and Borrelli [109, 111] use data from controller rollouts to learn a safe set and value function, with which recursive feasibility, stability, and local optimality can be

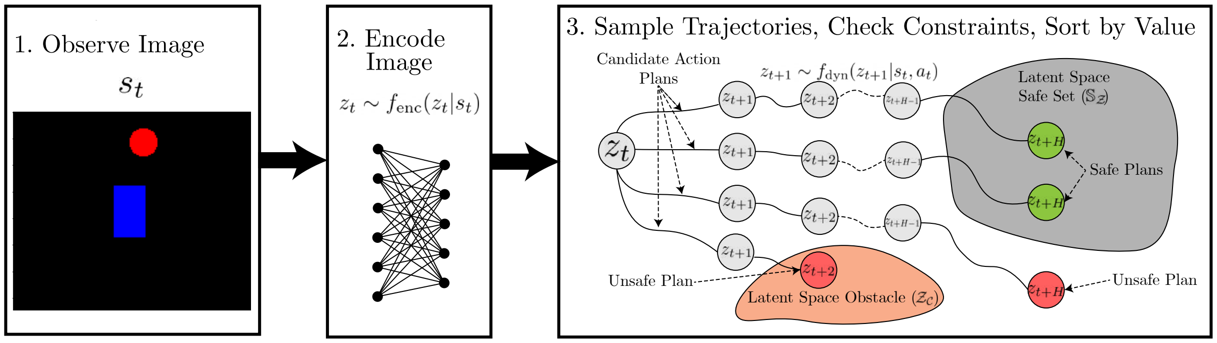

1. Observe Image
s_t
2. Encode Image
$z_t \sim f_{\text{enc}}(z_t|s_t)$
3. Sample Trajectories, Check Constraints, Sort by Value
Candidate Action Plans
$z_{t+1} \sim f_{\text{dyn}}(z_{t+1}|s_t, a_t)$
Latent Space Safe Set ($\mathbb{S}_\mathcal{Z}$)
z_t
z_{t+1}
z_{t+2}
z_{t+H-1}
z_{t+H}
Safe Plans
Unsafe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

Figure 7.1: **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 $z_t \sim f_{enc}(z_t|s_t)$. 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.

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. [106] 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, Berkenkamp, and Krause [175] 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.

7.1.2 Model Based Reinforcement Learning

There has been significant recent progress in algorithms which combine ideas from model-based planning and control with deep learning [138, 139, 140, 144, 86, 87]. These algorithms are gaining popularity in the robotics community as they enable leaning complex policies from data while maintaining some of the sample efficiency and safety benefits of classical model-based control techniques. However, these algorithms typically require hand-engineered dense cost functions for task specification, which can often be difficult to provide, especially in high-dimensional spaces. This motivates leveraging demonstrations (possibly suboptimal) to provide an initial signal regarding desirable agent behavior. There has been some prior work on leveraging demonstrations in model-based algorithms such as Quinlan and Khatib [149] and Ichnowski et al. [178], which use model-based control with known dynamics to refine initially suboptimal motion plans, and Fu, Levine, and Abbeel [140], which uses demonstrations to seed a learned dynamics model for fast online adaptation using iLQR [140]. Thananjeyan* et al. [106] and Zhu et al. [179] present ILC algorithms which rapidly improve upon suboptimal demonstrations when system dynamics are unknown. However, these algorithms either require knowledge of system dynamics [149, 178] or are limited to low-dimensional state spaces [140, 106, 179] and cannot be flexibly applied to visuomotor control tasks.

7.1.3 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 [162, 180]. 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 [181, 182, 171, 183, 173, 174, 162, 184, 180]. 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 [163, 180, 184] and goal-conditioned reinforcement learning [174, 171]. However, safe reinforcement learning for visuomotor tasks has received substantially less attention. Thananjeyan* et al. [46] and Kahn et al. [185] present reinforcement learning algorithms which estimate the likelihood of constraint violations to avoid them [46] or reduce the robot's velocity [185]. 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.

7.2 Problem Statement

We consider an agent interacting in a finite horizon goal-conditioned Markov Decision Processes (MDP) which can be described with the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{G}, \mathcal{A}, P(\cdot|\cdot,\cdot), R(\cdot,\cdot), \mu, T)$. $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ maps a state and action to a probability distribution over subsequent states, $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, μ is the initial state distribution ($s_0 \sim \mu$), and T is the time horizon. In this work, the agent is only provided with RGB image observations $s_t \in \mathbb{R}_+^{W \times H \times 3} = \mathcal{S}$, where W and H are the image width and height in pixels, respectively. We consider iterative tasks, where the agent must reach a fixed goal set $\mathcal{G} \subseteq \mathcal{S}$ as efficiently as possible and the support of μ is small. While there are a number of possible choices of reward functions that would encourage fast convergence to $\mathcal{G}$, providing shaped reward functions can be exceedingly challenging, especially when learning from high dimensional observations. Thus, as in Thananjeyan* et al. [106], 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 [46]. 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 $P^\pi_C(s)$ as the probability of future constraint violation (within time horizon T) under policy π from state s. The objective is to maximize the expected return R^π while maintaining a constraint violation probability lower than $\delta_\mathcal{C}$. This can be written formally as follows:

$$\pi^* = \underset{\pi \in \Pi}{\arg\max}\{R^\pi : \mathbb{E}_{s_0 \sim \mu}\left[P^\pi_C(s_0)\right] \leq \delta_\mathcal{C}\} \tag{7.1}$$

We assume that the agent is provided with an offline dataset $\mathcal{D}$ of transitions in the environment of which some subset $\mathcal{D}_{constraint} \subsetneq \mathcal{D}$ are constraint violating and some subset $\mathcal{D}_{success} \subsetneq \mathcal{D}$ appear in successful demonstrations from a suboptimal supervisor. As in [46], $\mathcal{D}_{constraint}$ 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.

7.3 Latent Space Safe Sets (LS^3)

We describe how LS^3 uses demonstrations and online interaction to safely learn iteratively improving policies. Section 7.3.1 describes how we learn a low-dimensional latent representation of image observations to facilitate efficient model-based planning. To enable this planning, we learn a probabilistic forward dynamics model as in [86] in the learned latent space and models to estimate whether plans will likely complete the task (Section 7.3.2) and to estimate future rewards and constraint violations (Section 7.3.3) from predicted trajectories. In Section 7.3.4, 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 7.3.2 and 7.3.3) after every K rollouts. See Algorithm 3 and the supplement for further details on training procedures and data collection.

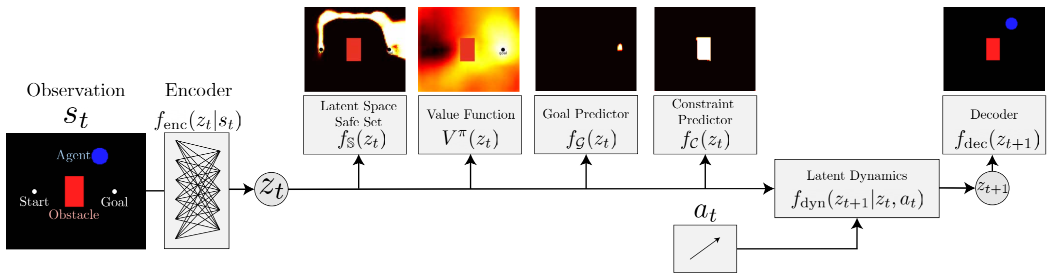


Figure 7.2: **LS^3 Learned Models**: LS^3 learns a low-dimensional latent representation of image-observations (Section 7.3.1) and learns a dynamics model, value function, reward function, constraint classifier, and safe set for constrained planning and task-completion driven exploration in this learned latent space. These models are then used for model-based planning to maximize the total value of predicted latent states (Section 7.3.3) while enforcing the safe set (Section 7.3.2) and user-specified constraints (Section 7.3.3).

Algorithm 3 Latent Space Safe Sets (LS^3)

```
Require: offline dataset $\mathcal{D}$, number of updates $U$
1: Train VAE encoder $f_{enc}$ and decoder $f_{dec}$ (Section 7.3.1) using data from $\mathcal{D}$
2: Train dynamics $f_{dyn}$, safe set classifier $f_{\mathbb{S}}$(Section 7.3.2), and the value function $V$ goal
   indicator $f_{\mathcal{G}}$, and constraint estimator $f_{\mathcal{C}}$ (Section 7.3.3) using data from $\mathcal{D}$.
3: for $j \in \{1, \ldots, U\}$ do
4:     for $k \in \{1, \ldots, K\}$ do
5:         Sample starting state $s_0$ from $\mu$.
6:         for $t \in \{1, \ldots, T\}$ do
7:             Choose and execute $a_t$ (Section 7.3.4)
8:             Observe $s_{t+1}$, reward $r_t$, constraint $c_t$.
9:             $\mathcal{D} := \mathcal{D} \cup \{(s_t, a_t, s_{t+1}, r_t, c_t)\}$
10:        end for
11:    end for
12:    Update $f_{dyn}$, $V$, $f_{\mathcal{G}}$, $f_{\mathcal{C}}$, and $f_{\mathbb{S}}$ with data from $\mathcal{D}$.
13: end for
```

7.3.1 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 [163, 184, 167, 186, 187, 180]. To learn such a representation, we train a β-variational autoencoder [188] on states in $\mathcal{D}$ to map states to a probability distribution over a d-dimensional latent space $\mathcal{Z}$. The resulting encoder network $f_{enc}(z|s)$ is then used to sample latent vectors $z_t \sim f_{enc}(z_t|s_t)$ 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. [189], 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 [163] and found that varying d did not significantly affect performance.

7.3.2 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 $\mathcal{D}_{success}$. This discrete set is difficult to plan to with continuous-valued state distributions so we leverage data from $\mathcal{D}_{success}$ (data in the sampled safe set), data from $\mathcal{D} \setminus \mathcal{D}_{success}$ (data outside the sampled safe set), and the learned encoder from Section 7.3.1 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: $\mathbb{S}^j_{\mathcal{Z}} = \{z_t | f_{\mathbb{S}}(\cdot)(z_t) \geq \delta_{\mathbb{S}}\}$.

7.3.3 Reward and Constraint Estimation

In this work, we define rewards based on whether the agent has reached a state $s \in \mathcal{G}$, but we need rewards that are defined on predictions from the dynamics, which may not correspond to valid real images. To address this, we train a classifier $f_{\mathcal{G}} : \mathcal{Z} \to \{0, 1\}$ to map the encoding of a state to whether the state is contained in $\mathcal{G}$ using terminal states in $\mathcal{D}_{success}$ (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 [106, 176, 111, 169], we train a recursively-defined value function (details in the supplement). Similar to the reward function, we use the encoder (Section 7.3.1) to train a classifier $f_{\mathcal{C}} : \mathcal{Z} \to [0, 1]$ with data of constraint violating states from $\mathcal{D}_{constraint}$ and the constraint satisfying states in $\mathcal{D} \setminus \mathcal{D}_{constraint}$ to map the encoding of a state to the probability of constraint violation.

7.3.4 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 7.1).

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:

$$\begin{aligned}
\underset{a_{t:t+H-1} \in \mathcal{A}^H}{\arg\max} \quad & \mathbb{E}_{z_{t:t+H}} \left[\sum_{i=1}^{H-1} f_{\mathcal{G}}(z_{t+i}) + V^{\pi}(z_{t+H}) \right] && (7.2) \\
\text{s.t.} \quad & z_t \sim f_{enc}(z_t | s_t) && (7.3) \\
& z_{k+1} \sim f_{dyn}(z_{k+1} | z_k, a_k) \; \forall k \in \{t, \dots, t+H-1\} && (7.4) \\
& \hat{\mathbb{P}}\left(z_{t+H} \in \mathbb{S}_{\mathcal{Z}}^{j-1}\right) \geq 1 - \delta_{\mathbb{S}} && (7.5) \\
& \hat{\mathbb{P}}(z_{t+i} \in \mathcal{Z}_{\mathcal{C}}) \leq \delta_{\mathcal{C}} \; \forall i \in \{0, \dots, H-1\} && (7.6)
\end{aligned}$$

In this problem, the expectations and probabilities are taken with respect to the learned, probabilistic dynamics model $f_{dyn}(z_{t+1}|z_t, a_t)$. The optimization problem is solved approximately using the cross-entropy method (CEM) [190] which is a popular optimizer in model-based RL [191, 106, 176, 192, 46].

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 7.2). First, the current state s_t is encoded to z_t (equation 7.3). 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 7.4). LS^3 constrains exploration using two chance constraints: (1) the terminal latent state in the plan must fall in the safe set (equation 7.5) and (2) all latent states in the trajectory must satisfy

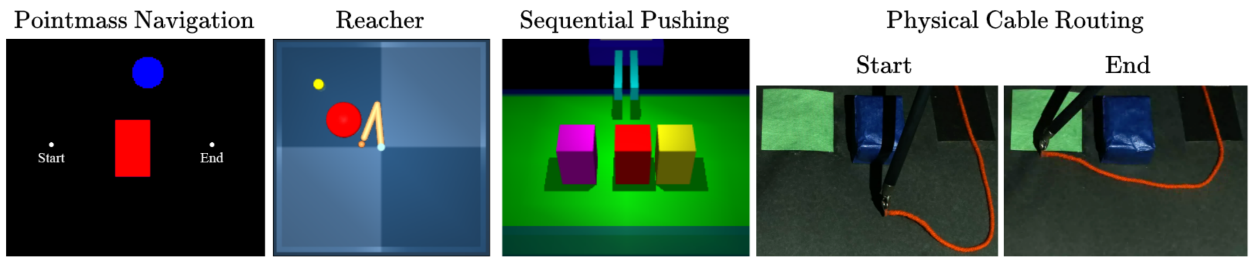


Figure 7.3: **Experimental Domains:** LS^3 is evaluated on 3 long-horizon, image-based, simulation environments: a visual navigation domain where the goal is to navigate the blue point mass to the right goal set while avoiding the red obstacle, a 2 degree of freedom reacher arm where the task is to navigate 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.

user-specified state-space constraints (equation 7.6). $\mathcal{Z}_\mathcal{C}$ is the set of all latent states such that the corresponding observation is constraint violating. The optimizer estimates constraint satisfaction probabilities for a candidate action sequence by simulating it repeatedly over f_{dyn}.

The first chance constraint ensures the agent maintains the ability to return to safe states where it knows how to do the task within H steps if necessary.

Because the agent replans at each timestep, the agent need not return to the safe set: during training, the safe set expands, enabling further exploration. In practice, we set $\delta_\mathbb{S}$ for the safe set classifier $f_\mathcal{S}$ adaptively as described in the supplement. The second chance constraint encourages constraint violation probability of no more than $\delta_\mathcal{C}$. After solving the optimization problem, the agent executes the first action in the plan: $\pi(z_t) = a_t$ where a_t is the first element of $a^*_{t:t+H-1}$, observes a new state, and replans.

7.4 Experiments

We evaluate LS^3 on 3 robotic control tasks in simulation and a physical cable routing task on the da Vinci Research Kit (dVRK) [105]. Safe RL is of particular interest for surgical robots such as the dVRK due to its delicate structure, motivating safety, and relatively imprecise controls [106, 142], 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.

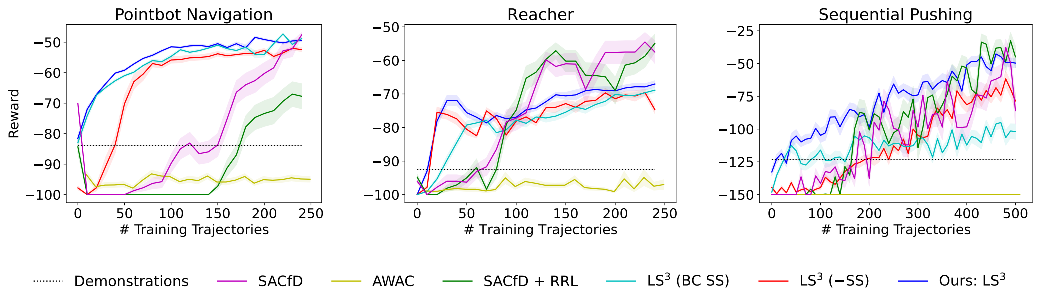


Figure 7.4: **Simulation Experiments Results:** Learning curves showing mean and standard error over 10 random seeds. We see that LS^3 learns more quickly than baselines and ablations. Although SACfD and SACfD+RRL converge to similar reward values, LS^3 is much more sample efficient and stable across random seeds.

7.4.1 Comparisons

We evaluate LS^3 in comparison to prior algorithms that behavior clone suboptimal demonstrations before exploring online (**SACfD**) [193] or leverage offline reinforcement learning to learn a policy using all offline data before updating the policy online (**AWAC**) [194]. For both of these comparisons we enforce constraints via a tuned reward penalty of λ for constraint violations as in [195]. We also implement a version of SACfD with a learned recovery policy (**SACfD+RRL**) using the Recovery RL algorithm [46] 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 7.5 (**LS^3 (−Safe 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 7.3.2) 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.

7.4.2 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 7.2, the total reward for a trajectory is the time to reach the goal set, where more negative rewards correspond to slower convergence to $\mathcal{G}$. Thus, for a task with task horizon T, a total reward greater than $-T$ implies successful task completion. The state is frozen in place upon constraint violation until the task horizon elapses. We also report task success and constraint satisfaction rates for LS^3 and comparisons during learning to study (1) the degree to which task completion influences sample efficiency and (2) how safely different algorithms explore. LS^3 collects $K = 10$ trajectories in between training phases on simulated tasks and $K = 5$ in between training phases for physical tasks, while the SACfD and AWAC comparisons update their parameters after each timestep. This presents a metric in terms of the amount of data collected across algorithms.

7.4.3 Domains

In simulation, we evaluate LS^3 on 3 vision-based continuous control domains that are illustrated in Figure 7.3. 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 7.2. We then study a constrained image-based reaching task (Reacher) based on [196], 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) [79] (Figure 7.3), 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.

7.4.4 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 7.4). 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 7.1 and Table 7.2, 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

Table 7.1: **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 7.2: **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

evaluated on long-horizon, image-based tasks as in this work, 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 7.3.2 (LS^3 (BC SS)) has similar performance to LS^3 on the navigation environment, but learns substantially more slowly on the Reacher environment and performs significantly worse than LS^3 on the more challenging Pushing environment as the learned safe set is unable to exploit temporal structure to distinguish safe states from unsafe states. See the supplement for details on experimental parameters and offline data used for LS^3 and comparisons and ablations studying the effect of the planning horizon and threshold used to define the safe set.

7.4.5 Physical Results

In physical experiments, we compare LS^3 to SACfD and SACfD+RRL (Figure 7.5) on the physical cable routing task illustrated in Figure 7.3. 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.

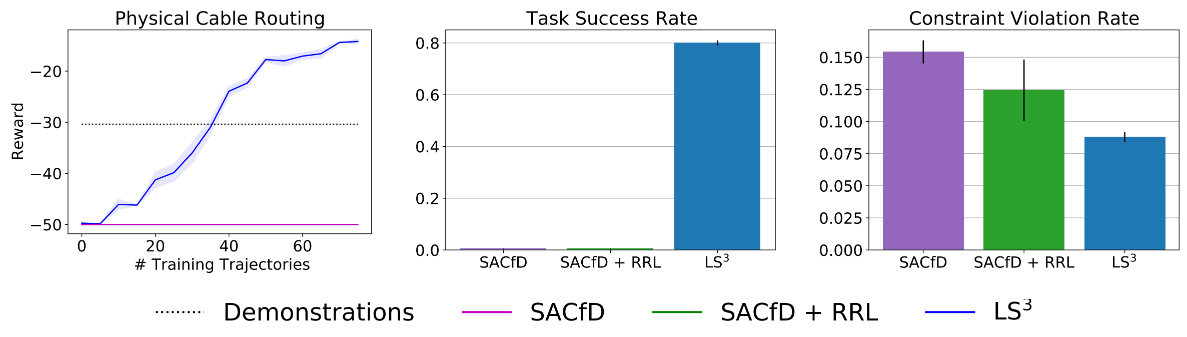


Figure 7.5: **Physical Cable Routing Results:** We present learning curves, task success rates and constraint violation rates with a mean and standard error across 3 random seeds. LS^3 learns a more efficient policy than the demonstrator while still violating constraints less than comparisons, which are unable to learn the task.

7.5 Discussion and Future Work

We present LS^3, a scalable algorithm for safe and efficient policy learning for visuomotor tasks. LS^3 structures exploration by learning a safe set in a learned latent space, which captures the set of states from which the agent is confident in task completion. LS^3 then ensures that the agent can plan back to states in the safe set, encouraging consistent task completion during learning. Experiments suggest that LS^3 can safely and efficiently learn 4 visuomotor control tasks, including a challenging sequential pushing task in simulation and a cable routing task on a physical robot. In future work, we are excited to explore further physical evaluation of LS^3 on safety critical visuomotor control tasks and applications to systems with dynamic constraints on velocity or acceleration.

Chapter 8

Monte Carlo Augmented Actor-Critic for Sparse Reward Deep RL from Suboptimal Demonstrations

Reinforcement learning has been successful in learning complex skills in a variety of environments [197, 198, 199], but providing dense, informative reward functions for RL agents is often very challenging to do accurately [200, 201, 202]. This is particularly challenging for high-dimensional control tasks, in which there may be a large number of factors that influence the agent's objective. In many settings, it may be much easier to provide sparse reward signals that simply convey high-level information about task progress, such as whether an agent has completed a task or has violated a constraint. However, optimizing RL policies given such reward signals can be exceedingly challenging, as sparse reward functions may not provide sufficient information to meaningfully distinguish between a wide range of different policies.

This issue can be mitigated by leveraging demonstrations, which provide initial signal about desired behaviors. Though demonstrations may often be suboptimal in practice, they should still serve to encourage exploration in promising regions of the state space, while allowing the agent to explore behaviors which may outperform the demonstrations. Prior work has considered a number of ways to leverage demonstrations to improve learning efficiency for reinforcement learning, including by initializing the policy to match the behavior of the demonstrator [203, 204], using demonstrations to explicitly constrain agent exploration [205, 206, 207], and introducing auxiliary losses to incorporate demonstration data into policy updates [208, 209, 210]. While these algorithms have shown impressive performance in improving the sample efficiency of RL algorithms, they add significant complexity and hyperparameters, making them difficult to implement and tune for different tasks.

We present Monte Carlo augmented Actor-Critic (MCAC), which introduces an easy-to-implement, but highly effective, change that can be readily applied to existing actor-critic algorithms without the introduction of any additional hyperparameters and only minimal additional complexity. The key idea is to encourage the RL agent to initially be optimistic in the neighborhood of successful trajectories, and progressively reduce this optimism during learning so that it can continue to explore new behaviors. To operationalize this idea, MCAC introduces two modifications to existing actor-critic algorithms. First, MCAC initializes the replay buffer with task demonstrations. Second, MCAC computes a modified target Q-value for critic updates by taking the maximum of the standard temporal distance (TD-1) targets and a Monte Carlo estimate of the reward-to-go. The intuition is that Monte Carlo value estimates can more effectively capture longer-term reward information, making it possible to rapidly propagate reward information from demonstrations through the learned Q-function. This makes it possible to prevent underestimation of values in high-performing trajectories early in learning, as eventual high rewards may be difficult to initially propagate back to earlier states with purely temporal distance targets [211]. Experiments on five continuous control domains suggest that MCAC is able to substantially accelerate exploration for both standard RL algorithms and recent RL from demonstrations algorithms in sparse reward tasks.

8.1 Related Work

8.1.1 Reinforcement Learning from Demonstrations

One standard approach for using demonstrations for RL first uses imitation learning [13] to pre-train a policy, and then fine-tunes this policy with on-policy reinforcement learning algorithms [212, 213, 204, 203]. However, initializing with suboptimal demonstrations can hinder learning and using demonstrations to initialize only a policy is inefficient, since this data can also be used for more accurate Q-value estimation.

Other approaches leverage demonstrations to explicitly constrain agent exploration. Thananjeyan et al. [205] and Wilcox et al. [207] propose model-based RL approaches that uses suboptimal demonstrations to iteratively improve performance by ensuring consistent task completion during learning. Similarly, Jing et al. [214] also uses suboptimal demonstrations to formulate a soft-constraint on exploration. However, a challenge with these approaches is that they introduce substantial algorithm complexity, making it difficult to tune and utilize these algorithms in practice. For example, while Thananjeyan et al. [205] does enable iterative improvement upon suboptimal demonstrations, it requires learning a model of system dynamics and a density estimator to capture the support of successful trajectories making it challenging to scale to high-dimensional observations.

Finally, many methods introduce auxiliary losses to incorporate demonstrations into policy updates Kim et al. [215], Gao et al. [210], and Kang, Jie, and Feng [216]. Deep Deterministic Policy Gradients from Demonstrations (DDPGfD) maintains all the demonstrations in a separate replay buffer and uses prioritized replay to allow reward information to propagate more efficiently Vecerik et al. [217], Nair et al. [208], and Hester et al. [209] use similar approaches, where demonstrations are maintained separately from the standard replay buffer and additional policy losses encourage imitating the behavior in the demonstrations. Mean-while, offline RL algorithms such as AWAC [218] use demonstration data to constrain the distribution of actions selected during online exploration. While these methods often work well in practice, they often increase algorithmic complexity and introduce several additional hyperparameters that are difficult and time consuming to tune, unlike MCAC which does neither and can easily be wrapped around any existing actor-critic algorithm.

8.1.2 Improving Q-Value Estimates

One of the core contributions of this work is to demonstrate an easy-to-implement, yet highly effective, method for stabilizing actor-critic methods for sparse reward tasks using demonstrations and an augmented Q-value target. There has been substantial literature investigating learning stability challenges in off-policy deep Q-learning and actor-critic algorithms. Please refer to Van Hasselt et al. [219] for a more thorough treatment of the learning stability challenges introduced by combining function approximation, bootstrapping, and off-policy learning, as well as prior work focused on mitigating these issues.

One class of approaches focuses on developing new ways to compute target Q-values [220, 211, 221, 222]. Van Hasselt, Guez, and Silver [221] computes target Q-values with two Q-networks, using one to select actions and the other to measure the value of selected actions, which helps to prevent the Q-value over-estimation commonly observed in practice in many practical applications of Q-learning. TD3 Fujimoto, Hoof, and Meger [62] attempts to address overestimation errors by taking the minimum of two separate Q-value estimates, but this can result in underestimation of the true Q-value target. Kuznetsov et al. [223] uses an ensemble of critics to adaptively address estimation errors in Q-value targets, but introduces a number of hyperparameters which must be tuned separately for different tasks. Wright et al. [224] and Schulman et al. [222] observe that different estimators of a policy's value make different bias and variance tradeoffs that can affect learning dynamics, which has some relationships to combining Monte Carlo and temporal difference based reward-to-go estimates as in MCAC.

Similar to our work, Wright et al. [211] explore the idea of taking a maximum over a bootstrapped target Q-value (TD-1 target) and a Monte Carlo estimate of the return-to-go to improve fitted Q-iteration. However, Wright et al. [211] does not use demonstrations and only considers simple low-dimensional control tasks using Q-learning with linear value approximation. To the best of our knowledge, MCAC is the first extension of these ideas to actor-critic algorithms with deep function approximation where we show surprising improvements in RL performance on complex high-dimensional continuous control tasks.

8.2 Problem Statement

We consider a Markov Decision Process (MDP) described by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, T)$ with a state set $\mathcal{S}$, an action set $\mathcal{A}$, a transition probability function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, a discount factor γ, and finite time horizon T. In each state

$s_t \in \mathcal{S}$ the agent chooses an action $a_t \in \mathcal{A}$ and observes the next state $s_{t+1} \sim p(\cdot|s_t, a_t)$ and a reward $r(s_t, a_t) \in \mathbb{R}$. The agent acts according to a policy π, which induces a probability distribution over $\mathcal{A}$ given the state, $\pi(a_t|s_t)$. The agent's goal is to find the policy π^* which at any given $s_t \in \mathcal{S}$ maximizes the expected discounted sum of rewards,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)\right], \tag{8.1}$$

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

In this work, we make additional assumptions specific to the class of problems we study. First, we assume that all transitions in the replay buffer are elements of complete trajectories; this is a reasonable assumption as long as all transitions are collected from rolling out some policy in the MDP. Second, we assume the agent has access to an offline dataset $\mathcal{D}_{\text{offline}}$ of (possibly suboptimal) task demonstrations. Finally, we focus on settings where the reward function is sparse, and takes on values in a small finite set. More formally, $r(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ where $|\mathcal{R}|$ is small (we use $|\mathcal{R}| \leq 4$ in all experiments). MCAC is not limited to sparse reward settings, but MDPs with unshaped rewards are often those in which MCAC is the most useful, as it can more efficiently direct exploration towards prior high performing trajectories.

8.3 Preliminaries: Actor-Critic Algorithms

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

$$Q^\pi(s_t, a_t) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k)\right]. \tag{8.2}$$

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

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

$$Q^{target}_{MC}(s_t, a_t) = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k), \tag{8.3}$$

and fitting Q^π_θ to these targets.

However, Q_{MC}^{target} is a very high variance estimator of the reward-to-go [222, 225], motivating the popular one-step temporal difference target (TD-1 target) to help stabilize learning:

$$Q_{TD}^{target}(s_t, a_t) = r(s_t, a_t) + \gamma Q_{\theta'}^{\pi}(s_{t+1}, a_{t+1}), \tag{8.4}$$

where $a_{t+1} \sim \pi_\phi(s_{t+1})$, which is recursively defined based on only a single (s_t, a_t, s_{t+1}, r_t) transition. Here θ' gives the parameters of a lagged target network [221], which is commonly used in value function learning with function approximation.

8.4 Monte Carlo augmented Actor-Critic

Here we discuss the algorithmic procedure of Monte Carlo augmented Actor-Critic. We first define the intuition behind MCAC in Section 8.4.1. Then, in Section 8.4.2, we describe how MCAC is implemented in practice.

8.4.1 MCAC Algorithm

The objective of MCAC is to efficiently convey information about sparse rewards from suboptimal demonstrations to Q_θ^π in order to accelerate policy learning while still maintaining learning stability. To do this, MCAC leverages two different methods of computing targets for fitting Q-functions to enable efficient value propagation throughout the state-action space while also learning a Q-value estimator with low enough variance for stable learning. To operationalize this idea, MCAC defines a new Q-function target for training Q_θ^π by taking the maximum of the Monte Carlo Q-target (eq 8.3) and the temporal difference Q-target (eq 8.4): $\max\left[Q_{\text{TD}}^{\text{target}}(s_t, a_t), Q_{\text{MC}}^{\text{target}}(s_t, a_t)\right]$.

The idea here is that early in learning, when the policy has not yet learned how to achieve high reward behaviors, a Q-function trained only with temporal difference targets will have very low values throughout the state-action space, as most online trajectories will be low performing. For sufficiently sparse rewards and long-horizon tasks, propagating information about delayed rewards through the temporal difference target can be very slow and sample inefficient [219]. On the other hand, the Monte Carlo Q-target can easily capture long-term rewards without a susceptibility to generalization issues in the learned Q-function [211], with the tradeoff that for unsuccessful trajectories it may dramatically underestimate Q-values. Thus, taking a maximum over these two targets serves to initially boost the Q-values for transitions near high performing trajectories while limiting the influence of underestimates from the Monte Carlo estimate.

Initially increasing the Q-values along demonstrator trajectories naturally biases early exploration towards more promising regions of the state space. Once MCAC starts sampling more high reward behaviors, the temporal distance targets are primarily trained on high performing trajectories, making it easier for them to capture information about sparse rewards and calibrate to the Monte Carlo targets. Once this is the case, the temporal distance targets learn reward-to-go estimates that are both accurate and low variance, making it possible for

Algorithm 4 Monte Carlo augmented Actor-Critic

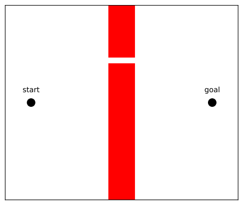
```
Require: Offline dataset $\mathcal{D}_{\text{offline}}$.
Require: Total training episodes $N$, batch size $M$
Require: Episode time horizon $T$.
 1: Initialize replay buffer $\mathcal{R} := \mathcal{D}_{\text{offline}}$.
 2: Initialize agent $\pi_\phi$ and critic $Q^\pi_\theta$.
 3: for $i \in \{1, \dots, N\}$ do
 4:     Initialize episode buffer $\mathcal{E} = \{\}$.
 5:     Observe state $s^i_1$.
 6:     for $j \in \{1, \dots, T\}$ do
 7:         Sample and execute $a^i_t \sim \pi_\theta(s^i_j)$.
 8:         Observe $s^i_{j+1}, r^i_j$.
 9:         $\tau^i_j \leftarrow (s^i_j, a^i_j, s^i_{j+1}, r^i_j)$
10:         $\mathcal{E} \leftarrow \mathcal{E} \cup \{\tau^i_j\}$.
11:         Sample $\mathcal{B} \subsetneq \mathcal{R}$ such that $|\mathcal{B}| = M$.
12:         Optimize $Q^\pi_\theta$ on $\mathcal{B}$ to minimize loss in eq (8.8).
13:         Optimize policy $\pi_\phi$ to maximize $Q_\theta$.
14:     end for
15:     for $\tau^i_j \in \mathcal{E}$ do
16:         Compute $Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j)$ as in eq (8.6).
17:         $\tau^i_j \leftarrow (s^i_j, a^i_j, s^i_{j+1}, r^i_j, Q^{\text{target}}_{\text{MC-}\infty}(s^i_j, a^i_j))$.
18:     end for
19:     $\mathcal{R} \leftarrow \mathcal{R} \cup \mathcal{E}$.
20: end for
```

MCAC to stably improve its performance. Notably, MCAC does not constrain its Q-targets explicitly based on the transitions in the demonstrations, making it possible for the policy to discover higher performing behaviors than those in the demonstrations.

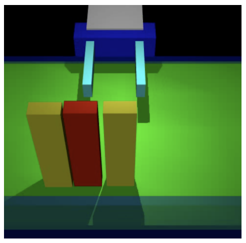
8.4.2 MCAC Practical Implementation

Here we discuss how MCAC can be implemented in practice. In principle, MCAC can be used in conjunction with any actor-critic RL algorithm, but in this work we specifically consider the application of MCAC to the Twin Delayed Deep Deterministic Policy Gradients (TD3) algorithm [62] and the Soft Actor-Critic (SAC) algorithm [193]. MCAC starts with an offline dataset of suboptimal demonstrations $\mathcal{D}_{\text{offline}}$, which are used to initialize a replay buffer $\mathcal{R}$. Then, during each episode i, we collect a full trajectory τ^i, where the j^{th} transition $(s^i_j, a^i_j, s^i_{j+1}, r^i_j)$ in τ^i is denoted by τ^i_j.

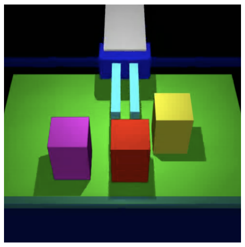
Next, consider any actor-critic method using a learned Q function approximator $Q_\theta(s_t, a_t)$ that, for a given transition $\tau^i_j = (s^i_j, a^i_j, s^i_{j+1}, r^i_j) \in \tau^i \subsetneq \mathcal{R}$, would be updated by minimizing



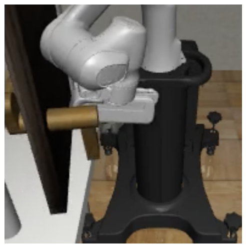
(a) Pointmass Nav



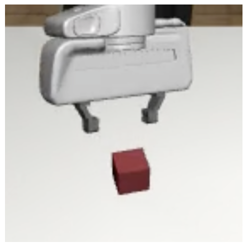
(b) Block Extraction



(c) Sequential Pushing



(d) Door Opening



(e) Block Lifting

Figure 8.1: We evaluate MCAC on five continuous control domains: a pointmass navigation environment, and four high-dimensional robotic control domains. All domains are associated with relatively unshaped reward functions, which only indicate constraint violation, task completion, or completion of a subtask based on the experimental domain.

a loss,

$$J(\theta) = \ell\left(Q_\theta(s_j^i, a_j^i), Q^{\text{target}}(s_j^i, a_j^i)\right), \tag{8.5}$$

where ℓ is an arbitrary differentiable loss function and Q^{target} is the target value for regressing Q_θ. To implement MCAC, we first calibrate with the TD-1 target (which provides infinite-horizon Q estimates) by computing the infinite horizon analogue of the Monte Carlo target defined in Equation 8.3, which assumes the last observed reward value will repeat forever and uses this to add an infinite sum of discounted rewards, and is given by

$$Q^{\text{target}}_{\text{MC-}\infty}(s_j^i, a_j^i) = \gamma^{T-j+1}\frac{r_T^i}{1-\gamma} + \sum_{k=j}^{T}\gamma^{k-j} r(s_k^i, a_k^i). \tag{8.6}$$

Then, we simply replace the target with a maximum over the original target and the Monte Carlo target defined in Equation 8.6, given by

$$Q^{\text{target}}_{\text{MCAC}}(s_j^i, a_j^i) = \max\left[Q^{\text{target}}(s_j^i, a_j^i), Q^{\text{target}}_{\text{MC-}\infty}(s_j^i, a_j^i)\right]. \tag{8.7}$$

This results in the following loss function for training Q_θ:

$$J(\theta) = \ell\left(Q_\theta(s_j^i, a_j^i), Q^{\text{target}}_{\text{MCAC}}(s_j^i, a_j^i)\right). \tag{8.8}$$

The full training procedure alternates between updating Q_θ^π using the method described above and optimizing the policy π_ϕ using the policy update method from the original algorithm MCAC is being applied to. The full MCAC algorithm is described in Algorithm 4.

8.5 Experiments

In the following experiments we study (1) whether MCAC enables more efficient learning when built on top of standard actor-critic RL algorithms, (2) whether MCAC can be applied to improve prior algorithms for RL from demonstrations, and (3) the sensitivity of MCAC to the quantity and quality of demonstrations and a slight modification in how target Q-values are constructed.

8.5.1 Experimental Procedure

We aggregate statistics over 10 random seeds for the experiments in Figure 8.2 and 5 seeds for all other experiments, reporting the mean and standard error across the seeds with exponential smoothing. Details on hyperparameters and implementation details are provided in Appendix G.1.

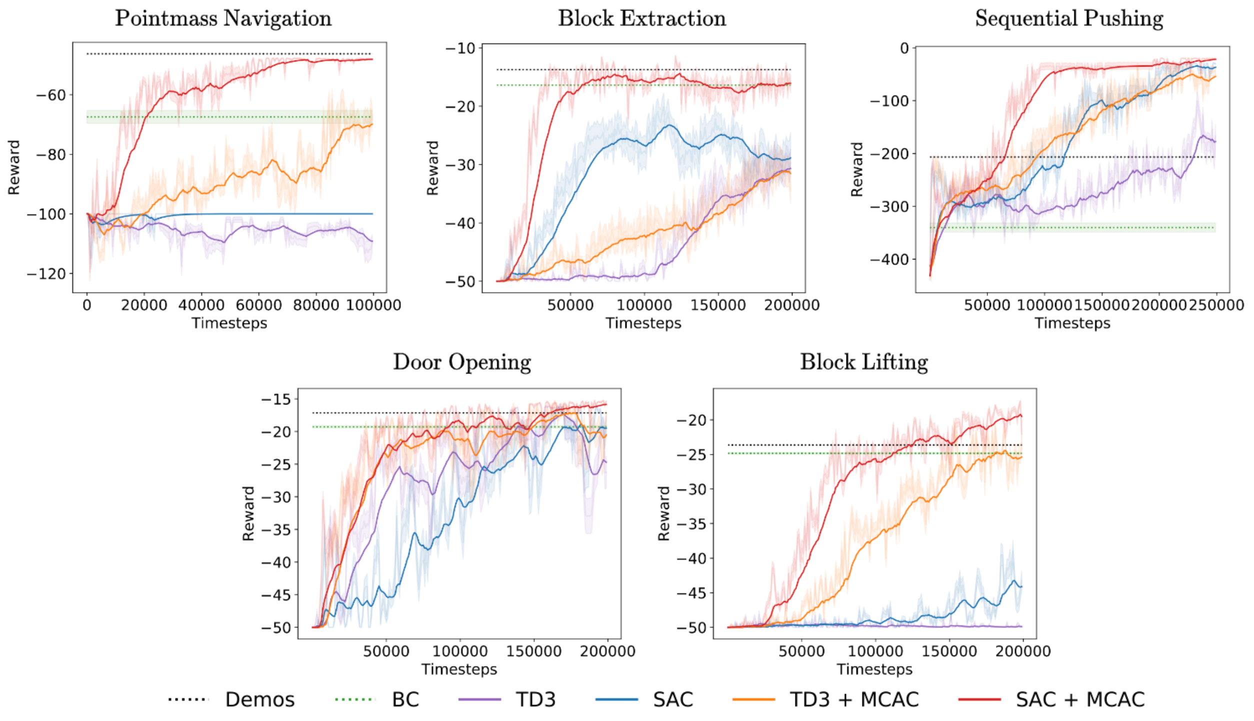


Figure 8.2: **MCAC and Standard RL Algorithms Results:** Learning curves showing the mean and standard error across 10 random seeds. Exponentially smoothed curves are shown in bold while raw curves are shown behind them. We find that MCAC improves the learning efficiency of both TD3 and SAC across all 5 environments.

8.5.2 Domains

We consider the five long-horizon continuous control tasks shown in Figure 8.1. All tasks follow the sparse reward assumption described in Section 8.2, making demonstrations critical for performance. We found that all of these domains posed significant exploration challenges for SAC and TD3 without demonstrations.

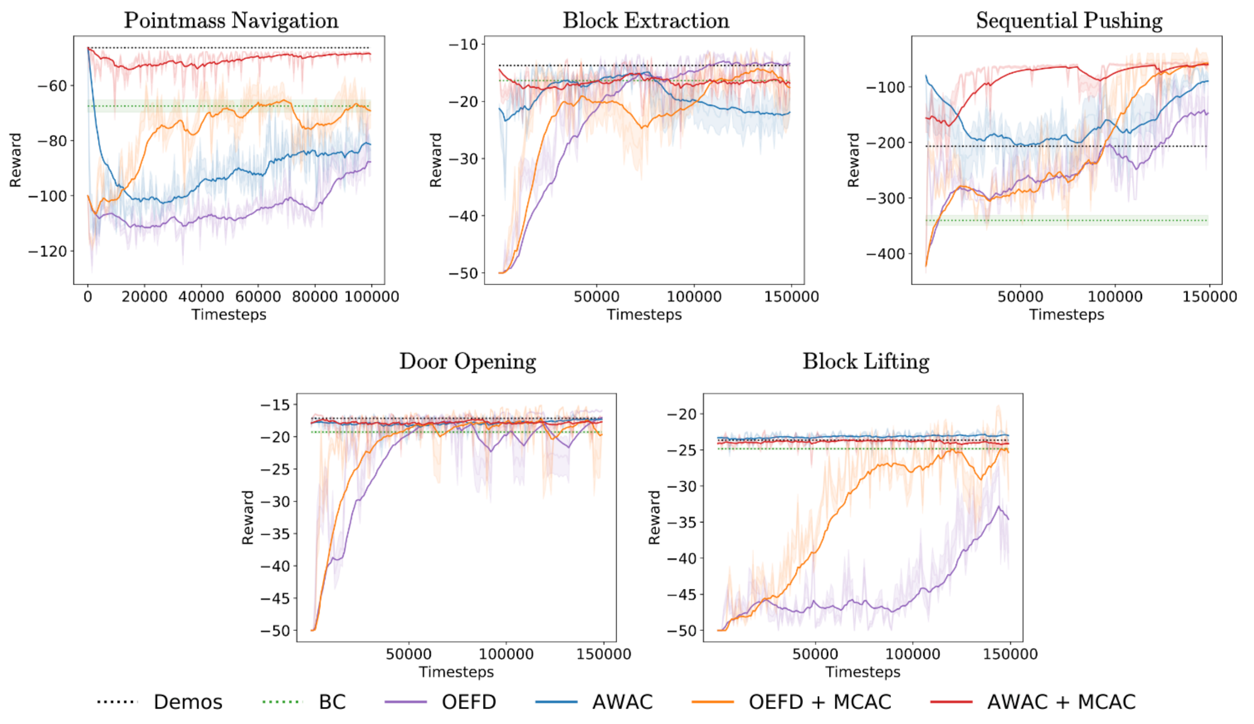
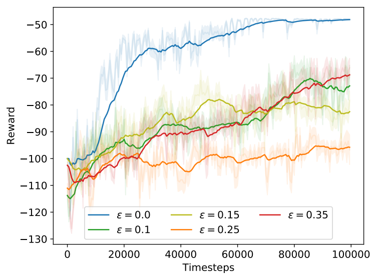


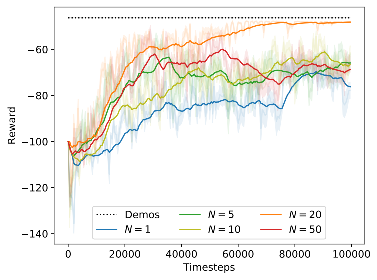
Figure 8.3: **MCAC and RL from Demonstrations Algorithm Results:** Learning curves showing the mean and standard error across 5 random seeds. Exponentially smoothed curves are shown in bold while raw curves are shown behind them. We find that in the environments in which OEFD or AWAC achieve high performance almost immediately, MCAC has little impact on performance. However, on the environments in which OEFD are unable to learn efficiently or have significant instability during learning, MCAC serves to significantly accelerate and stabilize policy learning.

Pointmass Navigation: The first domain is a pointmass 2D navigation task (Figure 8.1a) with time horizon $T = 100$, where the objective is to navigate around the red barrier from start set $\mathcal{S}$ to a goal set $\mathcal{G}$ by executing 2D delta-position controls. If the agent collides with the barrier it receives a reward of -100 and the episode terminates. To increase the difficulty of the task, we perturb the state with zero-mean Gaussian noise at each timestep. The combination of noisy transitions and sparse reward signal makes this a very difficult exploration task where the agent must learn to make it through the slit without converging to the local optima of avoiding both the barrier and the slit. At each time step, the agent receives a reward of -1 if it is not in the goal set and 0 if it is in the goal set.

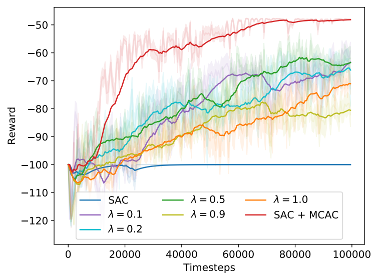
The demonstrator is implemented as a series of proportional controllers guiding it from the starting set to the slit, through the slit, and to the goal set. The actions are clipped to fit in the action space, and trajectories are nearly optimal. The learner is provided with 20 demonstrations.



(a) Demonstration Quality



(b) Demonstration Quantity



(c) Linear Combination Comparison

Figure 8.4: **MCAC Sensitivity Experiments:** Learning curves showing the mean and standard error across 5 random seeds for varying demonstration qualities (a) and quantities (b) for SAC + MCAC. Exponentially smoothed curves are shown in bold while raw curves are shown behind them. (a): Results suggest that MCAC is somewhat sensitive to demonstration quality, as when ϵ-greedy noise is injected into the demonstrator, MCAC's performance does drop significantly, although it eventually make some task progress for most values of ϵ. (b): MCAC appears to be much less sensitive to demonstration quantity, and is able to achieve relatively high performance even with a single task demonstration. (c): We also study how MCAC compares with simply defining Q targets with a weighted combination of the Monte Carlo returns (weight λ) and TD-1 returns (weight $1-\lambda$). We find that MCAC significantly outperforms all weightings, providing further evidence for the benefits of MCAC's parameter free method of taking the maximum of the Monte Carlo and TD-1 returns.

Object Manipulation in MuJoCo: We next consider two object manipulation tasks designed in the MuJoCo physics simulator [61], where the objective is to extract a block from a tight configuration on a table (Block Extraction, Figure 8.1b) and push each of a set of 3 blocks forward on the plane (Sequential Pushing, Figure 8.1c). In the Block Extraction task, the action space consists of 3D delta position controls and an extra action dimension to control the degree to which the gripper is opened. In the Sequential Pushing environment, this extra action dimension is omitted and the gripper is always kept closed. In the Block Extraction domain, the agent receives a reward of -1 for every timestep that it hasn't retrieved the red block and 0 when it has. In the Sequential Pushing domain, the reward increases by 1 for each block the agent pushes forward: the agent receives a reward of -3 when it has made no progress and 0 when it has completed the task. The Block Extraction task is adapted from Thananjeyan et al. [206] while the Sequential Pushing task is adapted from Wilcox et al. [207]. We use a time horizon of $T = 50$ for the Block Extraction task and $T = 150$ for the Sequential Pushing task.

The block extraction demonstrator is implemented as a series of proportional controllers guiding the arm to a position to grip the block, followed by an instruction to close the gripper and a controller to lift. We provide the agent with 50 slightly suboptimal demonstrations.

For the sequential pushing environment, the demonstrator uses a series of proportional controllers to slowly push one block forward, move backwards, line up with the next block, and repeat the motion until all blocks have been pushed. Because it moves slowly and moves far back from each block it pushes, demonstrations are very suboptimal. The learner is provided with 500 demonstrations.

Robosuite Object Manipulation: Finally, we consider two object manipulation tasks built on top of Robosuite [226], a collection of robot simulation tasks using the MuJoCo physics engine. We consider the Door Opening task (Figure 8.1d) and the Block Lifting task (Figure 8.1e). In the Door Opening task, a Panda robot with 7 DoF and a parallel-jaw gripper must turn the handle of a door in order to open it. The door's location is randomized at the start of each episode. At each timestep, the agent receives a reward of -1 if it has not opened the door and a reward of 0 if it has. In the Block Lifting task, the same Panda robot is placed in front of a table with a single block on its surface. The robot must pick up the block and lift it above a certain threshold height. The block's location is randomized at the start of each episode and the agent receives a reward of -1 for every timestep it has not lifted the block and a reward of 0 when it has. Both Robosuite tasks use a time horizon of $T = 50$.

For both Robosuite tasks, demonstrators are trained using SAC on a version of the task with a hand-designed dense reward function, as in the Robosuite benchmarking experiments Zhu et al. [226]. In order to ensure suboptimality, we stop training the demonstrator policy before convergence. For each Robosuite environment we use the trained demonstrator policies to generate 100 suboptimal demonstrations for training MCAC and the baselines.

8.5.3 Algorithm Comparisons

We empirically evaluate the following baselines both individually and in combination with MCAC. All methods are provided with the same demonstrations which are collected as described in Section 8.5.2. See Appendix G.1 for more in depth details on implementation and training.

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

Twin Delayed Deep Deterministic Policy Gradients (TD3) [62]: State of the art actor-critic algorithm which trains a deterministic policy to maximize a learned critic. In experiments we seed the replay buffer for TD3 with the same demonstrations used for MCAC.

Soft Actor-Critic (SAC) [193]: State of the art actor-critic algorithm which trains a stochastic policy which maximizes a combination of the Q value of the policy and the expected entropy of the policy to encourage exploration. In experiments we seed the replay buffer for SAC with the same demonstrations used for MCAC.

Overcoming Exploration from Demonstrations (OEFD) [208]: OEFD builds on DDPG [104] but adds an additional loss which encourages imitating demonstrations and a learned filter which determines when to activate this loss. We provide OEFD with the same demonstrations used for MCAC.

Advantage Weighted Actor-Critic (AWAC) [218]: A recent offline reinforcement learning algorithm designed for fast online fine-tuning. We also implement versions of each of the above RL algorithms with MCAC (TD3 + MCAC, SAC + MCAC, OEFD + MCAC, AWAC + MCAC). The behavior cloning comparison serves to determine whether online learning is beneficial in general, while the other comparisons study whether MCAC can be used to accelerate reinforcement learning for commonly used actor-critic algorithms (SAC and TD3) and also for recent algorithms for RL from demonstrations (OEFD and AWAC).

8.5.4 Results

In Section 8.5.4 we study whether MCAC enables more efficient learning when built on top of SAC and TD3, which are standard actor-critic RL algorithms widely used in the literature. Then, in Section 8.5.4 we study whether MCAC can provide similar benefits when applied to recent RL from demonstration algorithms (OEFD and AWAC). Finally, in Section 8.5.4, we investigate the sensitivity of MCAC to the quantity and quality of demonstrations on the pointmass environment, where demonstration quality can be easily controlled.

MCAC and Standard RL Algorithms

In Figure 8.2, we study the impact of augmenting SAC and TD3 with the MCAC target Q-function. Note that all methods, both with and without MCAC, initialize their replay buffers with the same set of demonstrations. The results in Figure 8.2 suggest that MCAC is able to significantly accelerate learning for both TD3 and SAC across all environments, and is able to converge to performance that is either on-par or better than the demonstrations. Furthermore, in the Pointmass Navigation and Block Lifting task, SAC and TD3 are unable to make much task progress without MCAC.

MCAC and RL From Demonstrations Algorithms

In Figure 8.3, we study the impact of augmenting OEFD Nair et al. [208] and AWAC Nair et al. [218] with the MCAC target Q-function. Results suggest that MCAC significantly improves the learning efficiency of OEFD on the Pointmass Navigation, Sequential Pushing, and Block Lifting tasks, but does not have a significant positive or negative affect on performance for the Block Extraction and Door Opening tasks. MCAC significantly improves the performance of AWAC on the Pointmass Navigation and Sequential Pushing environments, but does not significantly affect performance on the other 3 environments as AWAC is able to immediately converge to a stable policy after offline pre-training. These results suggest that for OEFD, MCAC gives a significant performance boost on 3/5 environments while not harming performance in the environments in which OEFD is already very effective. Similarly, in the Block Extraction, Door Opening, and Block Lifting environments, AWAC works almost immediately, and thus MCAC has no impact on its performance as desired. However, on the Pointmass Navigation and Sequential Pushing environments in which AWAC is unable to converge to a stable policy, MCAC significantly improves the stability of AWAC and enables it to converge to a high-performing policy.

MCAC Sensitivity Experiments

In Figure 8.4, we first study the impact of demonstration quality (Figure 8.4a) and quantity (Figure 8.4b) on MCAC when applied to SAC (SAC + MCAC) on the Pointmass Navigation domain. We evaluate sensitivity to demonstration quality by injecting ϵ-greedy noise into the demonstrator for the Pointmass Navigation domain. Results suggest that MCAC is somewhat sensitive to demonstration quality, since MCAC's performance does drop significantly for most values of ϵ, although it still typically makes some task progress. In Figure 8.4b, results suggest that MCAC is relatively robust to the number of demonstration.

We then compare MCAC with a modification in which Q-targets are computed via a weighted combination of Monte-Carlo and TD-1 targets with a weight parameter λ instead of by taking a maximum over the two, shown in Figure 8.4c. Results suggest that MCAC significantly outperforms this modification for a range of different weightings, providing further evidence for the benefits of MCAC in practice.

8.6 Discussion and Future Work

We present Monte Carlo augmented Actor-Critic (MCAC), a simple, yet highly effective, change that can be applied to any actor-critic algorithm in order to improve reinforcement learning from demonstrations. We present empirical results suggesting that MCAC often significantly improves, and does not significantly degrade, performance when applied to two state-of-the-art actor-critic RL algorithms and two RL from demonstrations algorithms on five different continuous control domains.

In future work, we hope to study Monte Carlo augmented Actor-Critic theoretically to better understand the exact mechanism behind why this simple change to standard actor critic algorithms can lead to such a significant increase in policy performance. We also will release an open-source implementation to encourage further research on algorithms for RL from demonstrations.

Part III

Reinforcement Learning from Negative Demonstrations

Chapter 9

Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones

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 [227], locomotion [228], and manipulation [74, 229]. 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-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

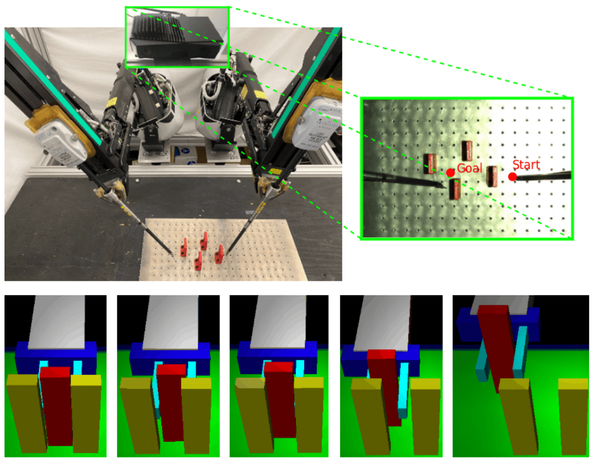


Figure 9.1: Recovery RL can safely learn policies for contact-rich tasks from high-dimensional image observations in simulation experiments and on a physical robotic system. We evaluate Recovery RL on an image-based obstacle avoidance task with delta-position control on the da Vinci Research Kit (top left) with overhead image observations (top right). We find that Recovery RL substantially outperforms prior methods (Figure 9.6), 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.

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 [230, 177, 231, 232], 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 [233], 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

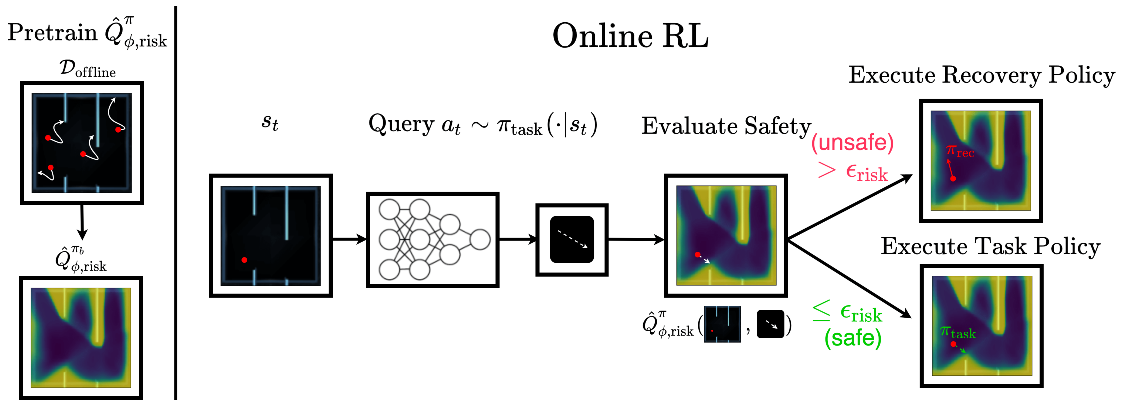

Pretrain $\hat{Q}^{\pi}_{\phi,\text{risk}}$
$\mathcal{D}_{\text{offline}}$
$\hat{Q}^{\pi_b}_{\phi,\text{risk}}$
Online RL
s_t
Query $a_t \sim \pi_{\text{task}}(\cdot|s_t)$
Evaluate Safety
$\hat{Q}^{\pi}_{\phi,\text{risk}}($, $)$
(unsafe)
$> \epsilon_{\text{risk}}$
Execute Recovery Policy
π_{rec}
Execute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

Figure 9.2: **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.

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.

9.1 Related Work

Prior work has studied safety in RL in several ways, including imposing constraints on expected return [155, 195], risk measures [234, 235, 236, 237], and avoiding regions of the MDP where constraint violations are likely [238, 239, 177, 106, 156, 240]. 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 [155], optimizing a Lagrangian relaxation [195, 241, 242], or constructing Lyapunov functions [243, 244]. The most similar of these works to Recovery RL is Srinivasan et al. [242], 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. [242], which uses the safety critic to modify the task policy optimization objective, Recovery RL uses it to determine when to execute a learned recovery policy which minimizes the safety critic to keep the agent in safe regions of the MDP. This idea enables Recovery RL to more effectively balance task performance and constraint satisfaction than algorithms which jointly optimize for task performance and safety.

Restricting Exploration with an Auxiliary Policy: Another approach to safe RL explicitly restricts policy exploration to a safe subset of the MDP using a recovery or shielding mechanism. This idea has been explored in [230, 177], which utilize Hamilton-Jacobi reachability analysis to define a task policy and safety controller, and in the context of shielding [231, 232, 245]. 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. [230] 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, Levine, and Abbeel [246] and Eysenbach et al. [238] 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 [238]. However, enforcing the ability to fully reset can be impractical or inefficient. Inspired by this work, Recovery RL instead executes approximate resets to nearby safe states when constraint violation is probable. Richter and Roy [227] 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. Rosolia and Borrelli [111] and Thananjeyan* et al. [176] introduce model predictive control algorithms which leverage initial constraint satisfying demonstrations to iteratively improve their performance with safety guarantees and Thananjeyan* et al. [106] 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 [106, 176] require a dynamics model to evaluate reachability-based safety online.

9.2 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:

$$\pi^* = \operatorname*{arg\,max}_{\pi \in \Pi} \{R^\pi : Q^\pi_{\text{risk}}(s_0, a_0) \le \epsilon_{\text{risk}}\} \tag{9.1}$$

This setting exactly corresponds to the CMDP formulation from [247], but with constraint costs limited to binary indicator functions for constraint violating states. We limit the choice to binary indicator functions, as they are easier to provide than shaped costs and use Q^π_{risk} to convey information about delayed constraint costs. We define the set of feasible policies, $\{\pi : Q^\pi_{\text{risk}} \le \epsilon\}$, the set of ϵ-safe policies Π_ϵ. Observe that if $\gamma_{\text{risk}} = 1$, then by the assumption of termination on constraint violation, $Q^\pi_{\text{risk}}(s_i, a_i) = \mathbb{P}\left(\bigcup_t C(s_t) = 1\right)$, or the probability of a constraint violation in the future. Setting $\epsilon_{\text{risk}} = 0$ as well results in a robust optimal control problem.

We present an algorithm to optimize equation (9.1) 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 $Q^\pi_{\text{risk}}(s, a) \le \epsilon_{\text{risk}}$. 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.

9.3 Recovery RL

We outline the central ideas behind Recovery RL. In Section 9.3.1, we review how to learn a safety critic to estimate the probability of future constraint violations for the agent's policy. Then in Section 9.3.2, 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 9.3.3 we discuss how the safety critic and recovery policy are initialized from offline data and in Section 9.3.4 we discuss implementation details. See Algorithm 5 and Figure 9.2 for further illustration of Recovery RL.

9.3.1 Preliminaries: Training a Safety Critic

As in Srinivasan et al. [242], Recovery RL learns a critic function Q^{π}_{risk} that estimates the discounted future probability of constraint violation of the current policy π:

$$\begin{aligned} Q^{\pi}_{\text{risk}}(s_t, a_t) &= \mathbb{E}_{\pi}\left[\sum_{t'=t}^{\infty} \gamma_{\text{risk}}^{t'-t} c_{t'} | s_t, a_t\right] \\ &= c_t + (1-c_t)\gamma_{\text{risk}}\mathbb{E}_{\pi}\left[Q^{\pi}_{\text{risk}}(s_{t+1}, a_{t+1}) | s_t, a_t\right]. \end{aligned} \tag{9.2}$$

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 9.2).

$$\begin{aligned} J_{\text{risk}}(s_t, a_t, s_{t+1}; \phi) = \frac{1}{2}\Big(&\hat{Q}^{\pi}_{\phi,\text{risk}}(s_t, a_t) - (c_t \\ &+ (1-c_t)\gamma_{\text{risk}} \mathop{\mathbb{E}}_{a_{t+1}\sim\pi(\cdot|s_{t+1})}[\hat{Q}^{\pi}_{\phi,\text{risk}}(s_{t+1}, a_{t+1})])\Big)^2 \end{aligned} \tag{9.3}$$

and use a target network to create the target values [242, 103].

9.3.2 Defining a Recovery Set and Policy

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

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

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:

$$a_t = \begin{cases} a_t^{\pi_{\text{task}}} & (s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{safe}} \\ a_t^{\pi_{\text{rec}}} & (s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{9.4}$$

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:

$$P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a) = \begin{cases} P(s'|s,a) & (s,a) \in \mathcal{T}^{\pi}_{\text{safe}} \\ P(s'|s,a^{\pi_{\text{rec}}}) & (s,a) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{9.5}$$

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 9.5? 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 5 Recovery RL

```
Require: $\mathcal{D}_{\text{offline}}$, task horizon $H$, number of episodes $N$
 1: Pretrain $\pi_{\text{rec}}$ and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on $\mathcal{D}_{\text{offline}}$ {Section 9.3.3}
 2: $\mathcal{D}_{\text{task}} \leftarrow \emptyset$, $\mathcal{D}_{\text{rec}} \leftarrow \mathcal{D}_{\text{offline}}$
 3: $s_0 \leftarrow$ env.reset()
 4: for $i \in \{1, \dots N\}$ do
 5:     for $t \in \{1, \dots H\}$ do
 6:         if $c_t = 1$ or is_terminal($s_t$) then
 7:             $s_t \leftarrow$ env.reset()
 8:         end if
 9:         $a_t^{\pi_{\text{task}}} \sim \pi_{\text{task}}(\cdot|s_t)$ {Query task policy}
10:         {Check if task policy will be unsafe}
11:         if $(s_t, a_t^{\pi_{\text{task}}}) \in \mathcal{T}^{\pi}_{\text{rec}}$ then
12:             $a_t \sim \pi_{\text{rec}}(\cdot|s_t)$ {Select recovery policy}
13:         else
14:             $a_t = a_t^{\pi_{\text{task}}}$ {Select task policy}
15:         end if
16:         Execute $a_t$
17:         Observe $s_{t+1}$, $r_t = R(s_t, a_t)$, $c_t = C(s_t)$
18:         {Relabel transition}
19:         $\mathcal{D}_{\text{task}} \leftarrow \mathcal{D}_{\text{task}} \cup \{(s_t, a_t^{\pi_{\text{task}}}, s_{t+1}, r_t)\}$
20:         $\mathcal{D}_{\text{rec}} \leftarrow \mathcal{D}_{\text{rec}} \cup \{(s_t, a_t, s_{t+1}, c_t)\}$
21:         Train $\pi_{\text{task}}$ on $\mathcal{D}_{\text{task}}$, $\pi_{\text{rec}}$ on $\mathcal{D}_{\text{rec}}$
22:         Train $\hat{Q}^{\pi}_{\phi,\text{risk}}$ on $\mathcal{D}_{\text{rec}}$ {Eq. 9.3}
23:     end for
24: end for
```

9.3.3 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 9.3 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 9.3.2 and illustrated in Algorithm 5. 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.

9.3.4 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 [104]. 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. [86] 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 [103] to learn π_{task}, but note that any RL algorithm could be used. Details on the implementation of both policies is in the supplement.

9.4 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 9.3) and an image-based obstacle avoidance task on a physical robot (Figure 9.6). 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) [105] 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 9.1 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 [248]. Furthermore, the dVRK system has been used in the past to evaluate safe RL algorithms [106] 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, 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.

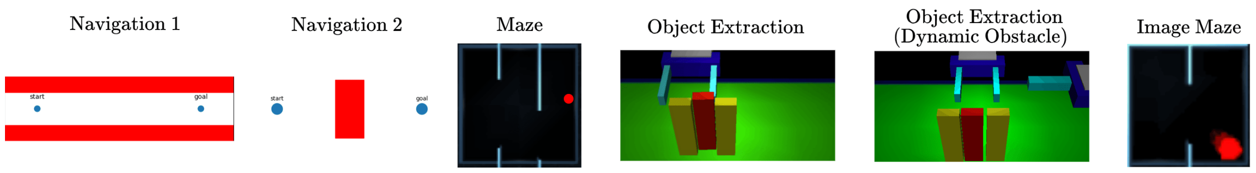


Figure 9.3: **Simulation Experiments Domains:** We evaluate Recovery RL on a set of 2D navigation tasks, two contact rich manipulation environments, and a visual navigation task. In Navigation 1 and 2, the goal is to navigate from the start set to the goal set without colliding into the obstacles (red) 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).

- **Safety Q-Functions for RL (SQRL)** [**242**]: 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)** [235]: 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)** [**195**]: 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 [103]) 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.

Results: We first study the performance of Recovery RL and prior methods in all simulation domains in Figure 9.4. Results suggest that Recovery RL with both model-free and model-based recovery mechanisms significantly outperform prior algorithms across

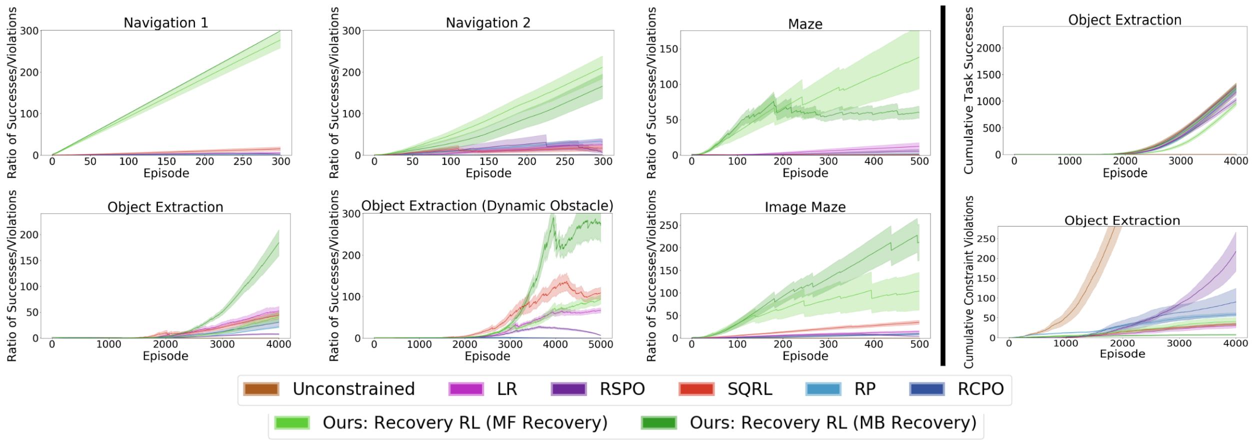


Figure 9.4: **Simulation Experiments:** *Left: ratio of successes to constraint violations over the course of online training.* In all navigation tasks, we find that Recovery RL significantly outperforms prior methods with both model-free and model-based recovery policies, while for the object extraction 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.

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 9.4, 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

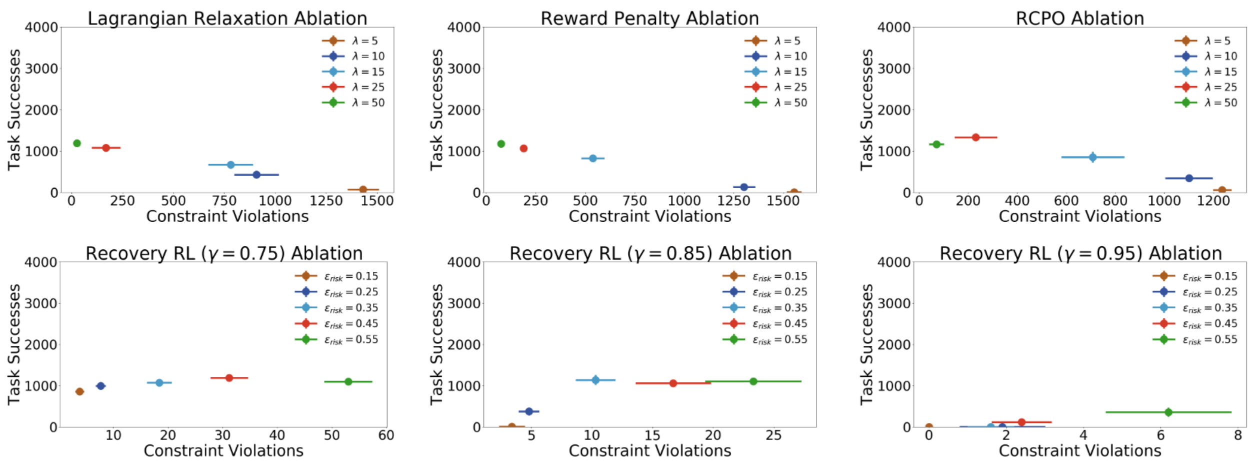


Figure 9.5: **Sensitivity Experiments:** We report the final number of task successes and constraint violations averaged over 10 runs at the end of training for Recovery RL and comparison algorithms for a variety of different hyperparameter settings on the Object Extraction task. We find that the comparison 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.

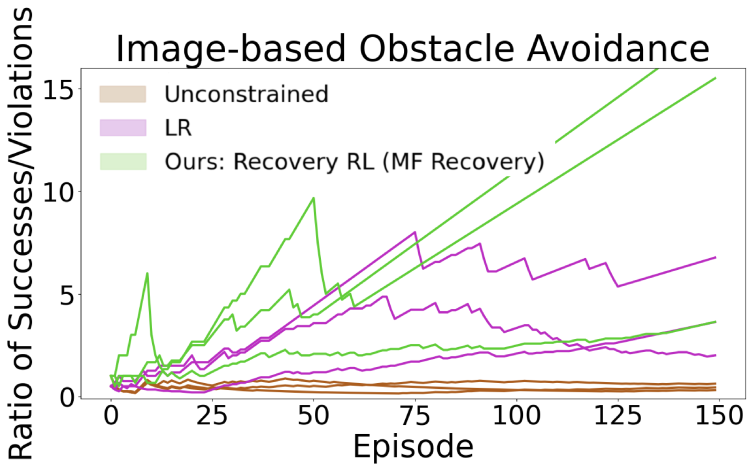
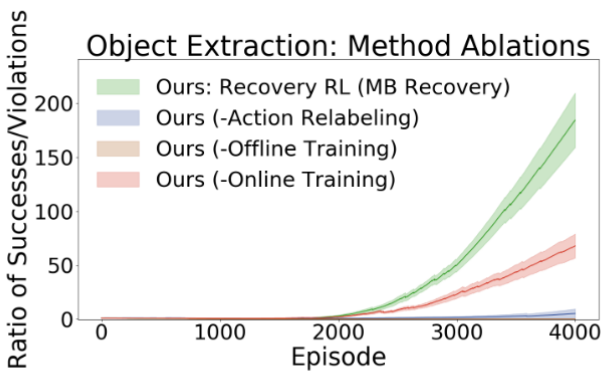


Figure 9.6: **Physical Experiment**: We evaluate Recovery RL on an image-based obstacle avoidance task (red obstacles) on the dVRK (Figure 9.1). We supply all algorithms with an overhead RGB image as input and run each algorithm 3 times. We find that Recovery RL significantly outperforms Unconstrained and LR.



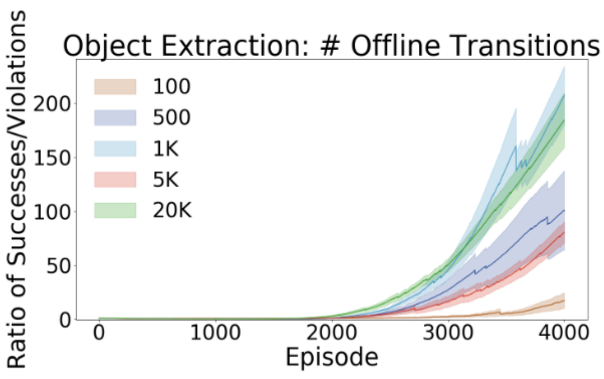


Figure 9.7: **Ablations:** We first study the affect of different algorithmic components of Recovery RL (left). Results suggest that offline pretraining of π_{rec} and $\hat{Q}^{\pi}_{\phi,\text{risk}}$ is critical for good performance, while removing online updates leads to a much smaller reduction in performance. Furthermore, we find that the action relabeling method for training π_{task} (Section 9.3.2) 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 9.1 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 9.6). 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 9.7. 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 9.3.2). 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 9.5, 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.

9.5 Discussion and Future Work

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 10

MESA: Offline Meta-RL for Safe Adaptation and Fault Tolerance

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 [46, 249, 242], 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 [242, 46, 250]. 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 [251] or wear-and-tear in manipulators or actuators [252]. These changes can drastically change the space of

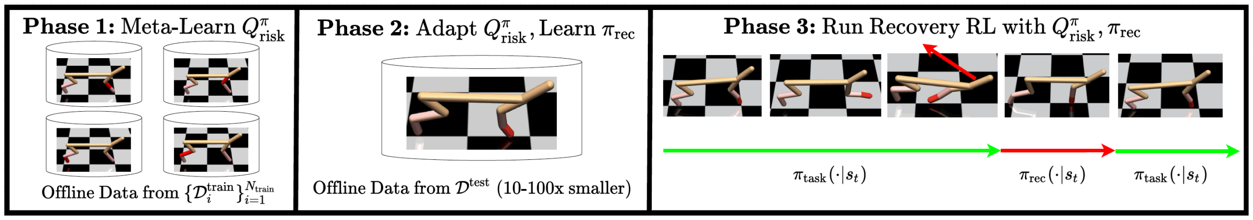

Phase 1: Meta-Learn Q^{π}_{risk}
Offline Data from $\{\mathcal{D}^{\text{train}}_i\}_{i=1}^{N_{\text{train}}}$
Phase 2: Adapt Q^{π}_{risk}, Learn π_{rec}
Offline Data from $\mathcal{D}^{\text{test}}$ (10-100x smaller)
Phase 3: Run Recovery RL with $Q^{\pi}_{\text{risk}}, \pi_{\text{rec}}$
$\pi_{\text{task}}(\cdot|s_t)$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

Figure 10.1: **Left: MEta-learning for Safe Adaptation (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.

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 work are (1) casting safe RL as an offline meta-reinforcement learning problem [253, 254], 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.

10.1 Related Work

10.1.1 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 [255]. Prior work has considered a number of methods for incorporating constraints into policy optimization for reinforcement learning, including trust region based methods [256, 155], optimizing a Lagrangian relaxation [195, 241, 242, 257], drawing connections to Lyapunov theory [243, 244, 156], anticipating violations with learned dynamics models [106, 176, 207, 250], using Gaussian processes to reason about uncertainty [258, 259], using recovery policies to shield the agent from constraint violations [231, 232, 246, 238, 46], formal reachability analysis [230, 177, 260, 261, 262, 263], or formal logic [245, 264]. Zhang et al. [250] 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. [250], 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. [242] 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. Srinivasan et al. [242], Bharadhwaj et al. [257], and Thananjeyan* et al. [46] present 3 different methods to utilize the learned safety critic for safe RL. Srinivasan et al. [242] and Thananjeyan* et al. [46] 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.

10.1.2 Meta Reinforcement Learning

There is a rich literature [265, 266, 267, 268, 269] studying learning agents that can efficiently adapt to new tasks. In the context of reinforcement learning, this problem, termed *meta-reinforcement learning* [270, 271, 272], 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 [270, 271, 273], gradient based optimization of policy parameters [272, 274], task inference [275, 276, 277], or adapting dynamics models for model-based RL [278, 279]. One of the core challenges studied in many meta-RL works is efficient exploration [280, 275, 281, 282], 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 [253, 254, 283] 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 work, 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 [284]. 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 [285], 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.

10.2 Preliminaries

10.2.1 Constrained Markov Decision Processes

In safe reinforcement learning, an agent interacts with a Constrained Markov Decision Process (CMDP) [247], defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, C, \rho_0, \gamma, \gamma_{risk})$, 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 $\gamma, \gamma_{risk} \in [0, 1]$ are the discount factors for the rewards and constraint values. As in prior work [242, 46], 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 $Q^\pi_{risk}(s_t, a_t) = \mathbb{E}_{\pi,\rho_0,P}\left[\sum_t^\infty \gamma^t_{risk} C(s_t)\right] = \mathbb{E}_{\pi,\rho_0,P}\left[\sum_t^\infty \gamma^t_{\text{risk}} \mathbb{P}\left(C\left(s_t\right) = 1\right)\right]$. Unlike unconstrained RL, safe RL agents seek to optimize:

$$\pi^* = \arg\max_{\pi} \left\{R^\pi : Q^\pi_{\text{risk}} \le \epsilon_{\text{risk}}\right\} \tag{10.1}$$

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

10.2.2 Safety Critics for Safe RL

Recent work investigates ways to estimate the discounted future probability of catastrophic constraint violation under the current policy: $Q^\pi_{\text{risk}}(s_t, a_t) = \sum_{t'=t}^\infty \gamma^{t'-t}_{risk} C(s_t)$ [46, 242]. In practice, algorithms search over a parametric function class: $\left\{Q^\pi_{\psi,risk}(s_t, a_t) : \psi \in \Psi\right\}$, 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:

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

where $Q^\pi_{\psi,risk,targ}$ 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 [195, 242, 257] with it or filtering dangerous actions [242, 46].

10.2.3 Recovery RL

In this work, 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 [46]. 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,

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

Here $\epsilon_{risk} \in [0, 1]$ 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}.

10.2.4 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 work 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) [272] optimizes the following objective at meta-train time:

$$\min_\theta \mathbb{E}_{\mathcal{M}_i\sim p(\mathcal{M})}\left[\mathcal{L}_{\mathcal{M}_i}(f_{\theta'_i})\right] = \min_\theta \mathbb{E}_{\mathcal{M}_i\sim p(\mathcal{M})}\left[\mathcal{L}_{\mathcal{M}_i}(f_{\theta-\alpha\nabla_\theta\mathcal{L}_{\mathcal{M}_i}(f_\theta)})\right] \tag{10.2}$$

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.

10.3 Problem Statement

We consider the offline meta-reinforcement learning problem setting introduced in [253, 254], 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 10.1.

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 10.1 in MDP $\mathcal{M}^{\text{test}}$ by learning some task τ in MDP $\mathcal{M}^{\text{test}}$ while minimizing constraint violations.

10.4 MEta-learning for Safe Adaptation (MESA)

We introduce MEta-learning for Safe Adaptation (MESA), a 3-phase procedure to optimize the objective in Section 10.3. First, MESA uses datasets of offline transitions from the training environments to meta-learn a safety critic optimized for rapid adaptation (Section 10.4.1). Then, we discuss how MESA adapts its meta-learned safety critic using a dataset of offline transitions from the test environment (Section 10.4.2). 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. [46], but we note that the learned safety critic can also be used in conjunction with other safe RL algorithms such as those from Srinivasan et al. [242] and Bharadhwaj et al. [257]. 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 10.4.3). The full algorithm is summarized in Algorithm 8 and Figure 10.1. An illustration of the safety critic adaptation procedure is shown in Figure 10.2.

10.4.1 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 $Q^{\pi}_{\psi,risk}$, with parameters ψ, using Model-agnostic Meta Learning [272]. We utilize the same safety critic loss function from [46]. The recovery policy is not trained with a MAML-style objective. Similar to the actor's loss function in DDPG [104], the recovery policy, parameterized by ω, aims to minimize the safety critic value for input state s_t:

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

10.4.2 Phase 2, Test Time Adaptation

A previously unseen test environment $\mathcal{M}^{test}$ 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 10.2.2) and $\mathcal{L}_{\pi_{rec}}(\omega, s)$ over $\mathcal{D}_{\text{test}}$ to rapidly adapt safety critic $Q^{\pi}_{\psi,risk}$ and train recovery policy $\pi_{\omega,rec}$

Note that the learned $Q^{\pi}_{\psi,risk}$ 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 $Q^{\pi}_{\psi,risk}$ serves as a pessimistic initialization for online learning of some downstream task τ. This is a desirable property, as $Q^{\pi}_{\psi,risk}$ will initially prevent constraint violations, and then become increasingly less pessimistic during online exploration when calibrated with the task policy for task τ.

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

We initialize the safety critic and recovery policy with the adapted $Q^{\pi}_{\psi,risk}$ 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 [46], both $Q^{\pi}_{\psi,risk}$ 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.

10.5 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. [103]. **Comparisons:**

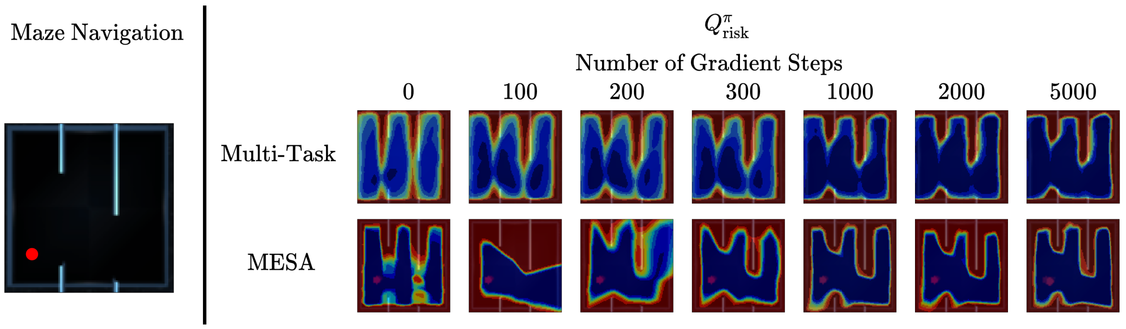
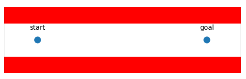
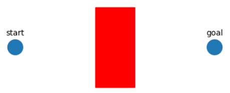


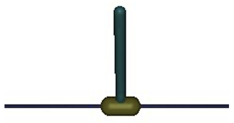
Figure 10.2: **Safety Critic Adaptation Visualizations:** For purposes of illustration, we evaluate MESA and a Multi-Task learning comparison on a simple Maze Navigation task (left) from [46] in which the objective is for the agent (the red dot) to navigate from a random point in the left column to the 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.



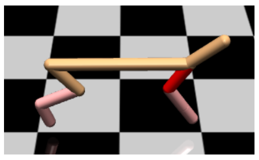
(a) Navigation 1



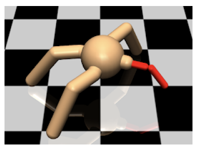
(b) Navigation 2



(c) Cartpole Length



(d) HalfCheetah Disabled



(e) Ant Disabled

Figure 10.3: **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.

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 [46]; **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 [46] 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 work.

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 10.3. 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 [46] 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 $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$.

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,

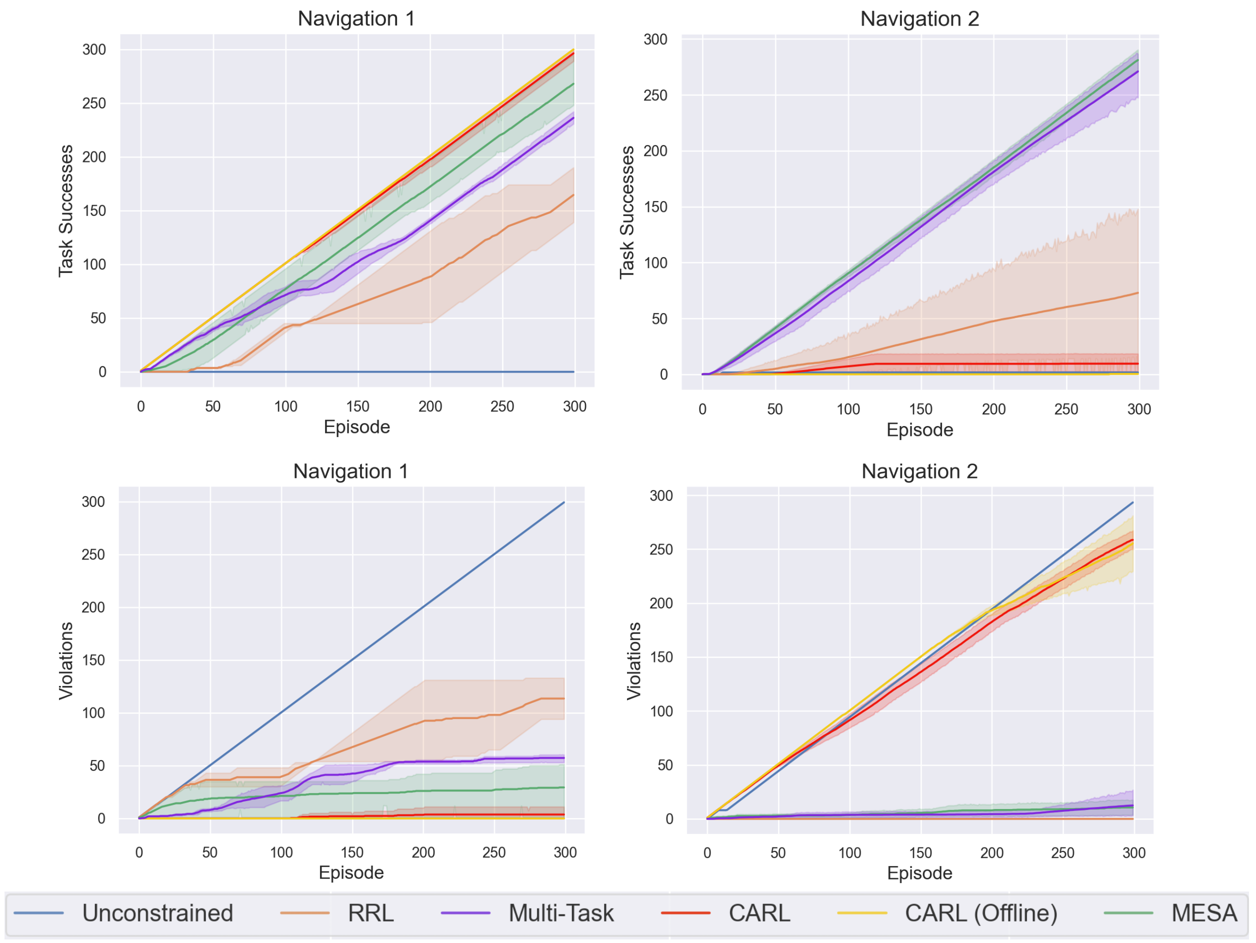


Figure 10.4: **Navigation Results: Top: Learning Curves (Phase 3).** MESA is able to achieve similar task success compared to prior algorithms on bot domains. **Bottom: Cumulative Constraint Violations (Phase 3).** Here, we find that MESA achieves fewer constraint violations than most comparisons, but find that the Multi-Task comparison also performs well on these environments.

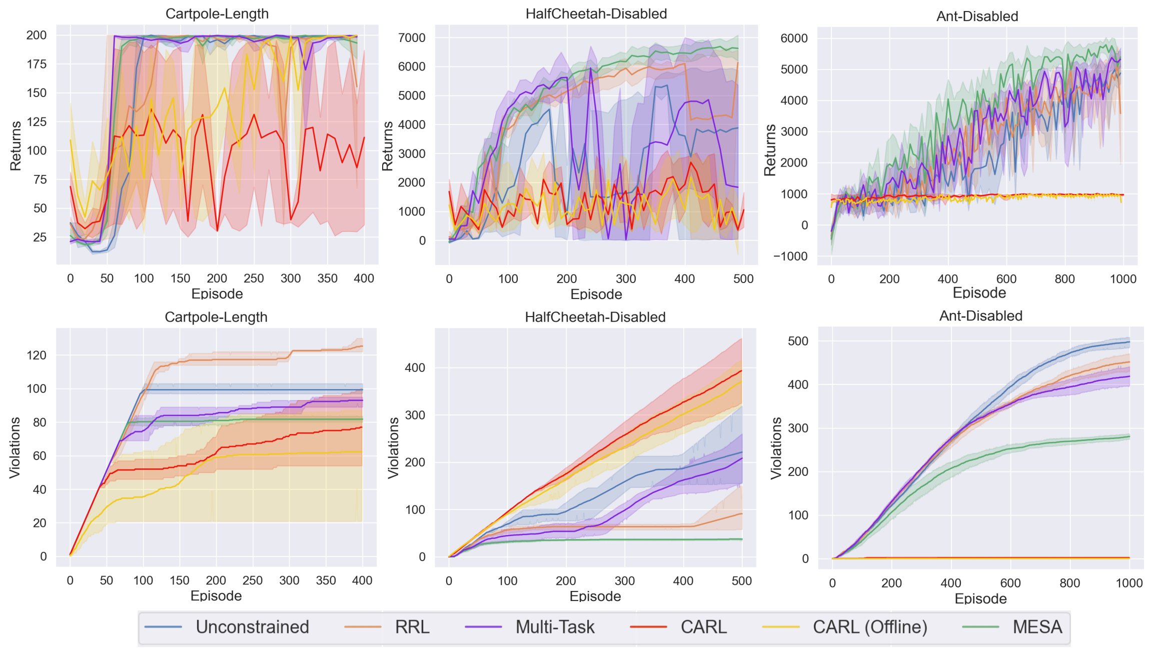


Figure 10.5: **Locomotion Results: Top: Learning Curves (Phase 3).** MESA achieves similar task performance as the best comparison algorithm, indicating that MESA is able to effectively learn in a test environment with previously unseen dynamics. **Bottom: Cumulative Constraint Violations (Phase 3).** 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.

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 [250] while the Ant-Disabled task is from [279].

10.5.1 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 [46]. 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.

10.5.2 Results

Navigation Results: We evaluate the performance of MESA and comparisons in Figure 10.4. 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 performance 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 10.5). 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.

10.6 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)?

10.6.1 Test Dataset Size

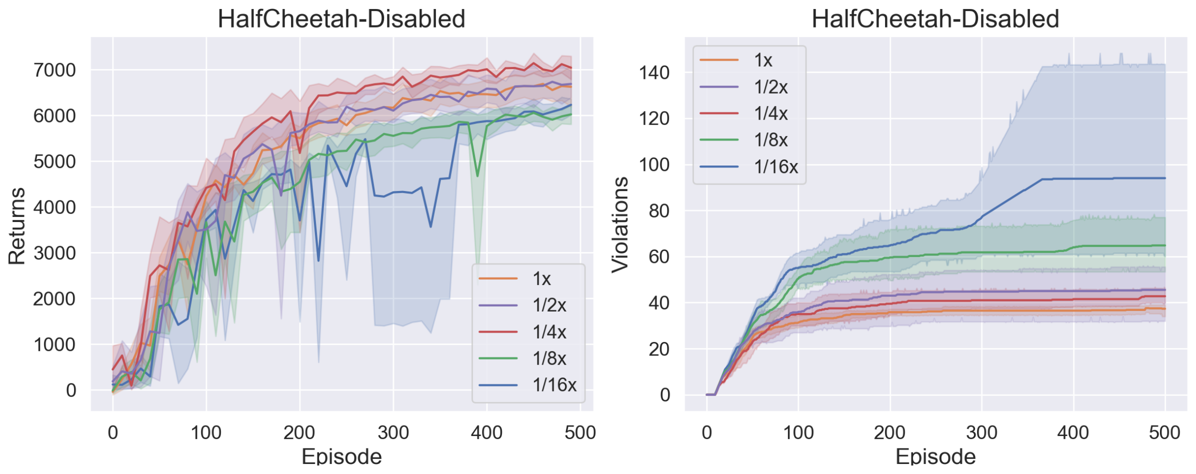
We first investigate the sensitivty of MESA to the size of the test dataset. Figure 10.6a, we study performance when the test dataset is 1x, 1/2x, 1/4x, 1/8x, and 1/16x the size of the test dataset (40K transitions) used for the HalfCheetah-Disabled results reported in Section 10.5. We find that MESA 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.

10.6.2 Test Environment Generalization

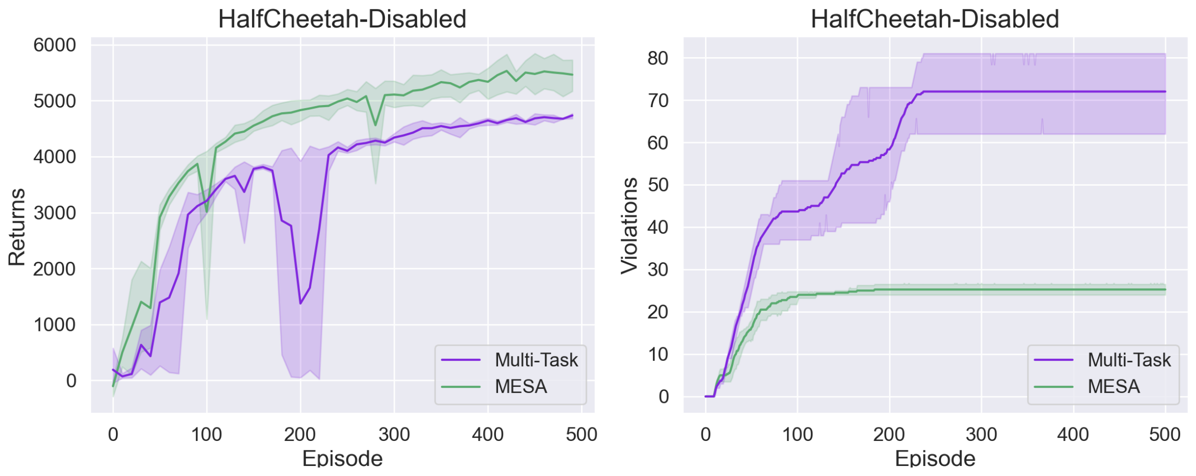
Here we study how MESA performs when the test environments have more significantly different dynamics from those seen during training. To evaluate this, we consider the HalfCheetah-Disabled task, and train MESA using the same training datasets considered in Section 10.5, in which specific joints are 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 10.6b, 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.

10.7 Discussion and Future Work

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. In future work we will explore applications of MESA to tasks on a physical robotic system.



(a) Varying Test Dataset Sizes



(b) Test Task Generalization: Partial Joint Failures

Figure 10.6: **Ablation**: **Sensitivity to Test Dataset Size:** In Figure 10.6a, 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 10.5, 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 10.6b, 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 10.5. 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.

Part IV

Learning Priors for Rapid Bandit-Based Grasp Exploration

Chapter 11

Accelerating Grasp Exploration by Leveraging Learned Priors

Robotic grasping has a wide range of industry applications such as warehouse order fulfillment, manufacturing, and assistive robotics. However, grasping is a difficult problem due to uncertainty in sensing and control, and there has been significant prior work on both analytical [286, 287, 288, 289, 290] and data-driven methods [291, 292, 293] for tackling these challenges. Recently, data-driven grasping algorithms have shown impressive success in learning grasping policies which generalize across a wide range of objects [294, 295, 296]. However, these techniques can fail to generalize to novel objects that are significantly different from those seen during training. Precisely, we investigate learning grasping policies for objects where general purpose grasping systems such as [294] produce relatively inaccurate grasp quality estimates, resulting in persistent failures during policy execution.

This motivates algorithms which can efficiently learn from on-policy experience by repeatedly attempting grasps on a new object and leveraging grasp outcomes to adjust the sampling distribution. Deep reinforcement learning has been a popular approach for online learning of grasping policies from raw visual input [297, 293, 173], but these approaches often take prohibitively long to learn robust grasping policies. These approaches typically attempt to learn *tabula rasa*, limiting learning efficiency. In this work, we introduce a method which leverages information from a general purpose grasping system to provide a prior for the learned policy while using geometric structure to inform online grasp exploration. We cast grasp exploration in the multi-armed bandits framework as in [298, 299]. However, unlike Laskey et al. [298] which focuses on grasping 2D objects where some rough geometric knowledge is known and Mahler et al. [299] which presents a method to transfer grasps to new 3D objects using a dataset of grasps on 3D objects with known geometry, we focus on efficiently learning grasping policies for 3D objects directly from depth image observations. In addition, the algorithm learns to grasp a specific object through online interaction, unlike Mahler et al. [299] which learns a general grasping policy for arbitrary objects. Specifically, we present a method which leverages prior grasp success probabilities from the state-of-the-art Dex-Net 4.0 grasp quality network GQ-CNN [294] to guide online grasp exploration on

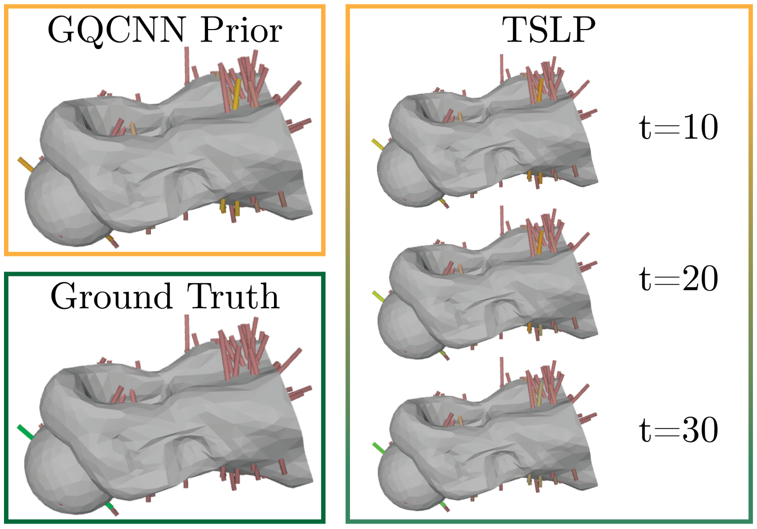


Figure 11.1: For adversarial objects, state-of-the-art grasp planning algorithms may incorrectly predict the distribution over grasp qualities (left column), where each whisker represents a grasp candidate colored by the likelihood of success (red indicates a poor grasp, green indicates a robust grasp). We find that TSLP can use the prior to efficiently discover the best grasp on the object (right column). Here, the policy discovers the only robust grasp despite a poor initial estimate of its quality from the GQ-CNN prior.

unknown 3D objects with only depth-image observations.

The contributions of this chapter are:

1. A new problem formulation for leveraging learned priors on grasp quality to accelerate online grasp exploration.

2. An efficient algorithm, Thompson Sampling with Learned Priors (TSLP), for learning grasping policies on novel 3D objects from depth images by leveraging priors from the Dex-Net 4.0 robot grasping system [300].

3. A new formulation of the mismatch between a prior distribution on grasp qualities and the ground truth grasp quality distribution and empirical analysis studying the effect of this mismatch on policy performance.

4. Simulation experiments suggesting that TSLP attains an average total reward 64.5% higher than a greedy baseline when evaluated over $300,000$ training runs across 3000 object poses and is able to effectively leverage information from a GQ-CNN prior.

11.1 Related Work

Robot grasping methods develop policies that execute grasps on novel objects, and can be divided into analytical methods and data-driven methods. Analytic methods assume knowledge of the geometry of the object to be grasped [288, 289, 287, 286, 301] or use geometric similarities between known and unknown objects to infer grasps on unknown objects [299] However, the generalization of these methods is limited for objects dissimilar to the known objects, or when geometric information is unknown [302], as in the case we consider. Data-driven methods rely on labels from humans [291, 303, 292, 296], self-supervision across many physical trials [297, 293, 173, 304], simulated grasp attempts [305, 306], or sim-to-real transfer methods such as domain randomization [307] or domain adaptation [295]. Hybrid approaches generate simulated grasp labels using analytical grasp metrics such as force closure or wrench resistance [300, 308, 294]. These data-driven and hybrid approaches train a deep neural network on the labeled data to predict grasp quality or directly plan reliable grasps on novel objects. A recent paper in sim-to-real transfer learning correct for inaccurate gripper poses predicted by the neural network by combining domain adaptation and visual servoing in the grasp planning process [309]. However, for adversarial objects [290], for which very few high quality grasps exist, or for objects significantly out of the training distribution, grasps planned by these methods may still fail. The presented method aims to leverage learned grasp quality estimates to enable efficient online learning for difficult-to-grasp objects through physical exploration of one pose of one object at a time, without previous knowledge of the object's geometry.

Past works have formulated grasp planning as a Multi-Armed Bandit problem for grasping 2D objects where some geometric knowledge is known [298] or for transferring grasps to unknown 3D objects using a dataset of grasps on 3D objects with known geometry. Laskey et al. [298] found that Thompson sampling with a uniform prior significantly outperformed uniform allocation or iterative pruning in 2D grasp planning in terms of convergence rate to within 3% of the optimal grasp, but their policy is limited to 2D grasps and cannot operate directly on visual inputs. Mahler et al. [299] extend [298] to 3D and incorporate prior information from Dex-Net 1.0, a dataset of over 10,000 3D object models and a set of associated robust grasps. The algorithm then uses Thompson sampling, in which the prior belief distribution for each grasp is calculated based on its similarity to grasps and objects from the Dex-Net 1.0 database [299]. For objects with geometrically similar neighbors in Dex-Net 1.0, the algorithm converges to the optimal grasp approximately 2 times faster than Thompson sampling without priors [299]. In contrast, we present a Bayesian multi-armed bandit algorithm for robotic grasping with depth image inputs that does not require a database to compute priors but instead leverages the Dex-Net 4.0 grasping system from [294] as a learned prior to guide active grasp exploration. Instead of learning a general grasping strategy for arbitrary objects as [299], the algorithm learns to grasp a specific object through online interactions with the object.

11.2 Problem Statement

Given a single unknown object on a planar workspace, the objective is to effectively leverage prior estimates on grasp qualities to learn a grasping policy that maximizes the likelihood of grasp success. We first define the parameters and assumptions on the environment (Sections 11.2.1 and 11.2.2), cast grasp exploration in the Bayesian bandits framework (Section 11.2.3), and formally define the policy learning objective (Section 11.2.4).

11.2.1 Assumptions

We make the following assumptions about the environment.

1. **Pose Consistency:** We assume that the object remains in the same pose during all rounds of learning. In simulation, this can be achieved by using ground-truth knowledge of physics and object geometry. In physical experiments, the pose consistency assumption will not hold generally. We discuss methods to approximately enforce pose consistency in physical experiments in Section 11.7.

2. **Evaluating Grasp Success:** We assume that the robot can evaluate whether a grasp has succeeded. In simulation, grasp success can be computed by using ground-truth knowledge of physics and object geometry. In physical experiments, success or failure can be determined using load cells, as in [294].

11.2.2 Definitions

1. **Observation:** An overhead depth image observation of the object at time $t = 0$ before policy learning has begun, given by $o \in \mathbb{R}_+^{H \times W}$.

2. **Arms:** We define a set of K arms, $\{a_k\}_{k=1}^K$.

3. **Actions:** Given a selected arm k we define a corresponding grasp action $u_k \in \mathcal{U}$.

4. **Reward Function:** Rewards for each arm are drawn from a Bernoulli distribution with unknown parameter p_k: $r(u_k) \sim \text{Ber}(p_k)$. Here $r(u_k) = 1$ if executing u_k results in the object being successfully grasped, and 0 otherwise.

5. **Priors:** We assume access to priors on the Bernoulli parameter p_k for each arm k.

6. **Policy:** Let $\pi_\theta(u_k)$ denote a policy parameterized by θ which selects an arm k and executes the action u_k. Thus, $\pi_\theta(u_k)$ defines a distribution over $\mathcal{U}$ at any given timestep t.

11.2.3 Bayesian Bandits

A multi-armed bandits problem is defined by an agent which must make a decision at each timestep $t \in \{1, 2, \dots T\}$ by selecting an arm $k \in \{1, 2, \dots K\}$ to pull. After each arm pull, the agent receives a reward which is sampled from an unknown reward distribution. In the Bayesian bandits framework [310], the agent maintains a belief over the parameters of the reward distribution for each arm, which can optionally be seeded with a known prior. The objective is to learn a policy with a distribution over arms that maximizes the cumulative expected reward over T rounds.

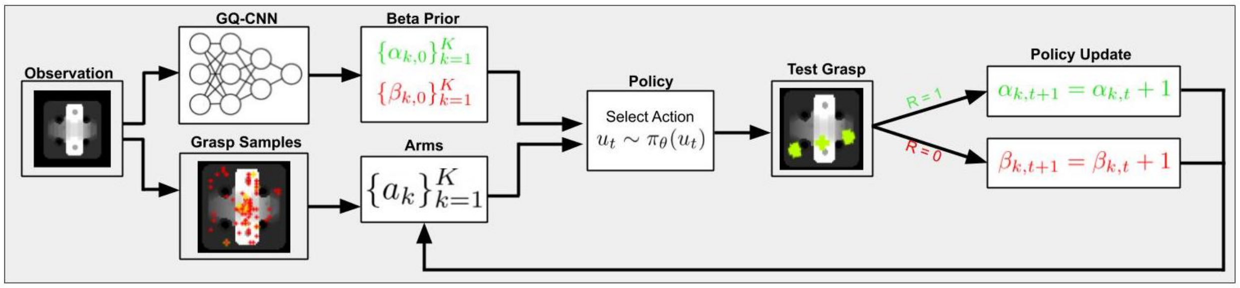

Observation
GQ-CNN
Grasp Samples
Beta Prior
$\{\alpha_{k,0}\}_{k=1}^{K}$
$\{\beta_{k,0}\}_{k=1}^{K}$
Arms
$\{a_k\}_{k=1}^{K}$
Policy
Select Action
$u_t \sim \pi_\theta(u_t)$
Test Grasp
R=1
R=0
Policy Update
$\alpha_{k,t+1} = \alpha_{k,t} + 1$
$\beta_{k,t+1} = \beta_{k,t} + 1$

Figure 11.2: **Method Overview:** A pre-trained GQ-CNN is used to set the priors on the reward parameters for each arm given the initial observation o and arms are sampled on observation o. Then, at each timestep the learned policy selects an arm and executes the corresponding action in the environment. The Thompson sampling parameters are updated based on the reward received as described in Section 11.3.1.

11.2.4 Learning Objective

The objective in policy learning is to maximize the total accumulated reward, which corresponds to maximizing the frequency with which the object is grasped. Let u_t denote the action selected at timestep t. Then the objective is to learn policy parameters θ to maximize the following:

$$J(\theta) = \mathbb{E}_{u_t \sim \pi_\theta(u_t)} \left[\sum_{t=1}^{T} r(u_t) \right] \tag{11.1}$$

11.3 Grasp Exploration Method

We discuss how to leverage learned priors from GQ-CNN to guide grasp exploration by using Thompson sampling, to learn a vision-based grasping policy. Since rewards are drawn from a Bernoulli distribution as defined in Section 11.2, we represent the prior with a Beta distribution, the conjugate prior for a Bernoulli distribution. As noted in [298], this choice of prior is convenient since we can update the belief distribution over an arm k after executing corresponding action u_k in closed form given the sampled reward. See Figure 11.2 for a full method overview.

11.3.1 Thompson Sampling with a Beta-Bernoulli Process

Given that we pull arm k at time t and receive reward $r(u_k) \in \{0, 1\}$, as shown in [298], we can form the posterior of the Beta distribution by updating the shape parameters $\alpha_{k,t}$ and $\beta_{k,t}$:

$$\begin{aligned} \alpha_{k,t+1} &= \alpha_{k,t} + r(u_k) \\ \beta_{k,t+1} &= \beta_{k,t} + (1 - r(u_k)) \end{aligned}$$

For Thompson sampling, at time t, the policy samples $\hat{p}_{k,t} \sim \text{Beta}(\alpha_{k,t}, \beta_{k,t})$ for all arms $k \in \{1, 2, \ldots K\}$, selects arm $k^* = \arg\max_k \hat{p}_{k,t}$, and executes the corresponding action u_{k^*} in the environment. Note that the expected Bernoulli parameter for arm k can be computed from the current shape parameters $\alpha_{k,t}$ and $\beta_{k,t}$ as follows:

$$\mathbb{E}\left[\hat{p}_{k,t}\right] = \frac{\alpha_{k,t}}{\alpha_{k,t} + \beta_{k,t}} \tag{11.2}$$

However, it remains to appropriately initialize $\alpha_{k,0}$ and $\beta_{k,0}$. Note that setting $\alpha_{k,0} = \beta_{k,0} = 1 \ \forall \ k \in \{1, 2, \ldots K\}$ corresponds to a prior which is uniform on $[0, 1]$ for Bernoulli parameter $p_{k,t}$. We instead set $\alpha_{k,0}, \beta_{k,0}$ according to a learned prior by using the initial depth image observation o.

11.3.2 Leveraging Neural Network Priors

We use a pre-trained Grasp Quality Convolutional Neural Network (GQ-CNN) from [294] to obtain an initial estimate of the probability of grasp success. GQ-CNN learns a Q-function, $Q_\phi(\cdot, \cdot)$, which given an overhead depth image of an object and a proposed parallel jaw grasp, estimates the probability of grasp success. However, as explored in [290], there exist many objects for which the analytical methods used for training GQ-CNN are relatively inaccurate, resulting in significant errors. Thus, we refine the initial GQ-CNN grasp quality estimates with online exploration.

We first compute $Q_\phi(o, u_k) \ \forall \ k \in \{1, 2, \ldots K\}$ and use these estimates as each arm's initial mean Bernoulli parameter. Note that $\alpha_{k,t}$ and $\beta_{k,t}$, as defined in Section 11.3.1, correspond to the cumulative number of grasp successes and grasp failures respectively for action u_k up to time t. Thus, $(\alpha_{k,0}, \beta_{k,0})$ can be interpreted as pseudo-counts of grasp successes and failures respectively for action u_k before policy learning has begun, while prior strength $S = \alpha_{k,0} + \beta_{k,0}$ can be interpreted as the number of pseudo-rounds before policy learning. If S is large, the prior induced by $(\alpha_{k,0}, \beta_{k,0})$ will significantly influence the expected Bernoulli parameter given in 11.2 for many rounds, while if S is small, the resulting prior will be quickly washed out by samples from online exploration. We enforce the following initial conditions for $(\alpha_{k,0}, \beta_{k,0})$, given the GQ-CNN prior:

$$\frac{\alpha_{k,0}}{\alpha_{k,0} + \beta_{k,0}} = Q_\phi(o, u_k)$$
$$\frac{\beta_{k,0}}{\alpha_{k,0} + \beta_{k,0}} = 1 - Q_\phi(o, u_k)$$

For a desired prior strength $S = \alpha_{k,0} + \beta_{k,0}$, we set:

$$\alpha_{k,0} = S \cdot Q_\phi(o, u_k)$$
$$\beta_{k,0} = S \cdot (1 - Q_\phi(o, u_k))$$

This prior enforcement technique in conjunction with online learning with Thompson Sampling, as discussed in Section 11.3.1, results in a stochastic policy $\pi_\theta(u_k)$ parameterized by $\theta = \left(\{(\alpha_k, \beta_k)\}_{k=1}^K, \phi\right)$, the learned Beta distribution shape parameters across all arms and the fixed parameters of the GQ-CNN used for initialization.

11.3.3 Prior Mismatch

To measure the quality of the GQ-CNN prior, we define a notion of dissimilarity between the prior and ground truth grasp probabilities, as in Chapelle and Li [311], termed the *prior mismatch*. However, unlike Chapelle and Li [311], which primarily focuses on mismatch between the mean of the prior distribution and true Bernoulli parameter, we present a new metric based on the discrepancy between how arms are ranked under the prior and under the ground truth distribution.

Given the grasp quality estimates of the GQ-CNN prior $q_p = (Q_\phi(o, u_k))_{k=1}^K$ and the ground truth grasp probabilities $q_g = (p_k)_{k=1}^K$ on all K arms, let $\mathcal{P} = \{(q_p[k], q_g[k])\}_{k=1}^K$. We then compute Kendall's tau coefficient, defined as:

$$\tau = \frac{N_c - N_d}{\sqrt{(N_c + N_d + T_p)(N_c + N_d + T_g)}}$$

where N_c and N_d are the number of concordant and discordant pairs in $\mathcal{P}$, respectively, and T_p and T_g are the number of pairs for which $q_p[i] = q_p[j]$ and $q_g[i] = q_g[j]$, respectively [312, 313]. As a rank correlation coefficient, $\tau \in [-1, 1]$, where 1 denotes a perfect match in the rankings and -1 denotes perfectly inverse rankings. We define the prior mismatch M as a dissimilarity measure that maps τ to $[0, 1]$:

$$M = \frac{1 - \tau}{2}$$

In practice, to control for stochasticity when sampling arms on the initial observation o, we average M over 10 independently sampled sets of K arms.

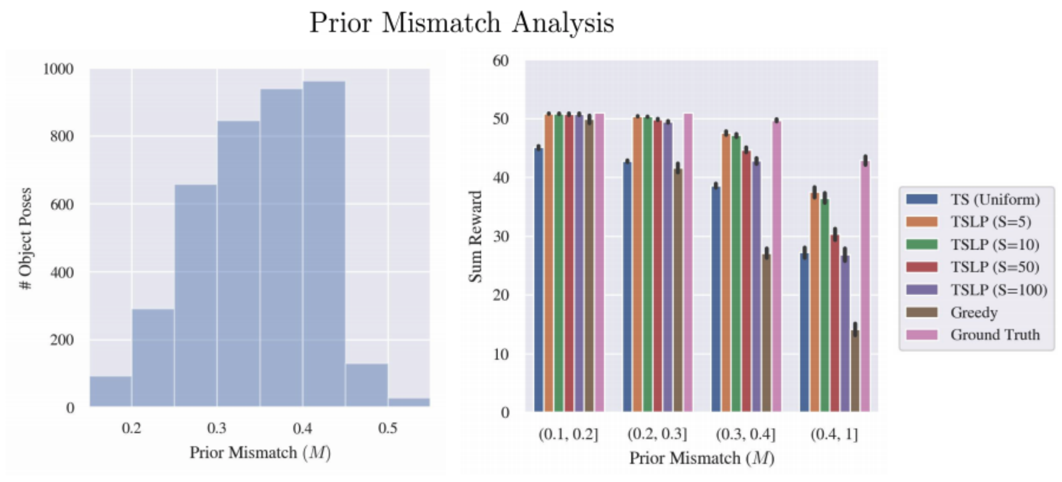
11.4 Practical Implementation

We implement the method from Section 11.3 in a simulated environment using 3D object models from the Dex-Net 4.0 dataset [294]. We render a simulated depth image of the object using camera parameters that are selected to be consistent with a Photoneo PhoXi S industrial depth camera. Arms are selected by sampling parallel-jaw antipodal grasp candidates on the observation o using the antipodal image grasp sampling technique from Dex-Net 2.0 [300]. The antipodal grasp sampler thresholds the depth image to find areas with high gradients, then uses rejection sampling over pairs of pixels to find antipodal grasp points. Each parallel jaw grasp is represented by a center point $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ and a grasp axis $\mathbf{v} \in \mathbb{R}^3$ [299]. They are visualized as whiskers in Figures 11.1 and 11.5. Once the arms are sampled from the image, we calculate the prior grasp probabilities using GQ-CNN, then deproject each grasp from image space into a 3D grasp using the known camera intrinsics. Note that TSLP can also be easily be applied with different types of grasps such as Suction grasps [308] provided that the actions corresponding to the arms are parameterized accordingly. We then iteratively choose grasps according to the policy for a set number of timesteps and collect the reward for each grasp.

Algorithm 6 Thompson Sampling with Learned Priors (TSLP) for Image-Space Grasp Exploration

```
Input: Number of arms ($K$), Maximum Iterations $T$, Pretrained GQ-CNN $Q_\phi(\cdot,\cdot)$, Prior Strength $S$
Output: Grasp exploration policy: $\pi_\theta(u_k)$
  Capture observation $o$, sample $K$ antipodal grasps $\{a_k\}_{k=1}^{K}$, and compute prior beliefs $\alpha_{k,0}, \beta_{k,0}\ \forall\ k \in \{1, 2, \dots K\}$ using $Q_\phi(o, u_k)$ using method from Section 11.3.2.
  for $t = 1, ..., T$ do
    Select action $u_k$ using Thompson sampling as in Section 11.3.1
    Execute $u_k$ and observe $r(u_k)$
    Update $\alpha_{k,t}, \beta_{k,t}\ \forall k \in \{1, 2, \dots K\}$ as in Section 11.3.1
  end for
```

Algorithm 6 summarizes the full approach discussed in Section 11.3 along with implementation details. If we are unable to sample K arms or if none of the corresponding grasps has ground truth quality greater than zero, we do not consider the object pose. In simulation, we evaluate the probability of grasp success for each arm using the robust wrench resistance metric, which measures the grasp's ability to resist the gravity wrench under perturbations in the grasp pose, as in [308]. Then, rewards during policy learning and evaluation are sampled from a Bernoulli distribution with parameter defined by this metric. Note that while computing this metric requires knowledge of the object geometry, this metric is simply used to simulate grasp success on a physical robotic system and is not exposed to TSLP.



(a) (b)

Figure 11.3: (a) The distribution of prior mismatch M for the 3946 total object poses in the dataset used by [294]. We find that M ranges from 0.16 to 0.64 and a value between 0.4 and 0.45 is the most common, accounting for about 25% of the object poses. All object poses with M above 0.55 are placed into the highest bin. (b) The sum of rewards over policy evaluation computed over 3000 object poses randomly selected from the dataset. This plot suggests that the chosen metric accurately describes the mismatch between the prior and ground truth grasp quality distribution, as performance of all TSLP policies decreases with increased prior mismatch.

11.5 Experiments

11.5.1 Setup

In simulation experiments, we evaluate both the accuracy of the prior mismatch metric and the ability of TSLP to increase grasp exploration efficiency. We assess whether TSLP can discover higher quality grasps than baselines which do not explore online or which explore online but do not leverage learned priors for the grasp selection policy. In both experiments, we make use of the dataset from Mahler et al. [294], which contains approximately 1,600 object meshes.

We evaluate the learned policies every 10 steps of learning, and perform 500 learning steps in total for all experiments. To evaluate the learned policies, we sample 100 grasps from the current policy without policy updates and compute the metric defined in Equation (11.1). We evaluate TSLP with a variety of different prior strengths to evaluate how important the GQ-CNN prior is for policy performance. We also compare to Thompson sampling with a uniform prior over the arms. Thus, this policy does not utilize the GQ-CNN prior at all, and all learning is performed online. Note that when evaluating policies, there are two key

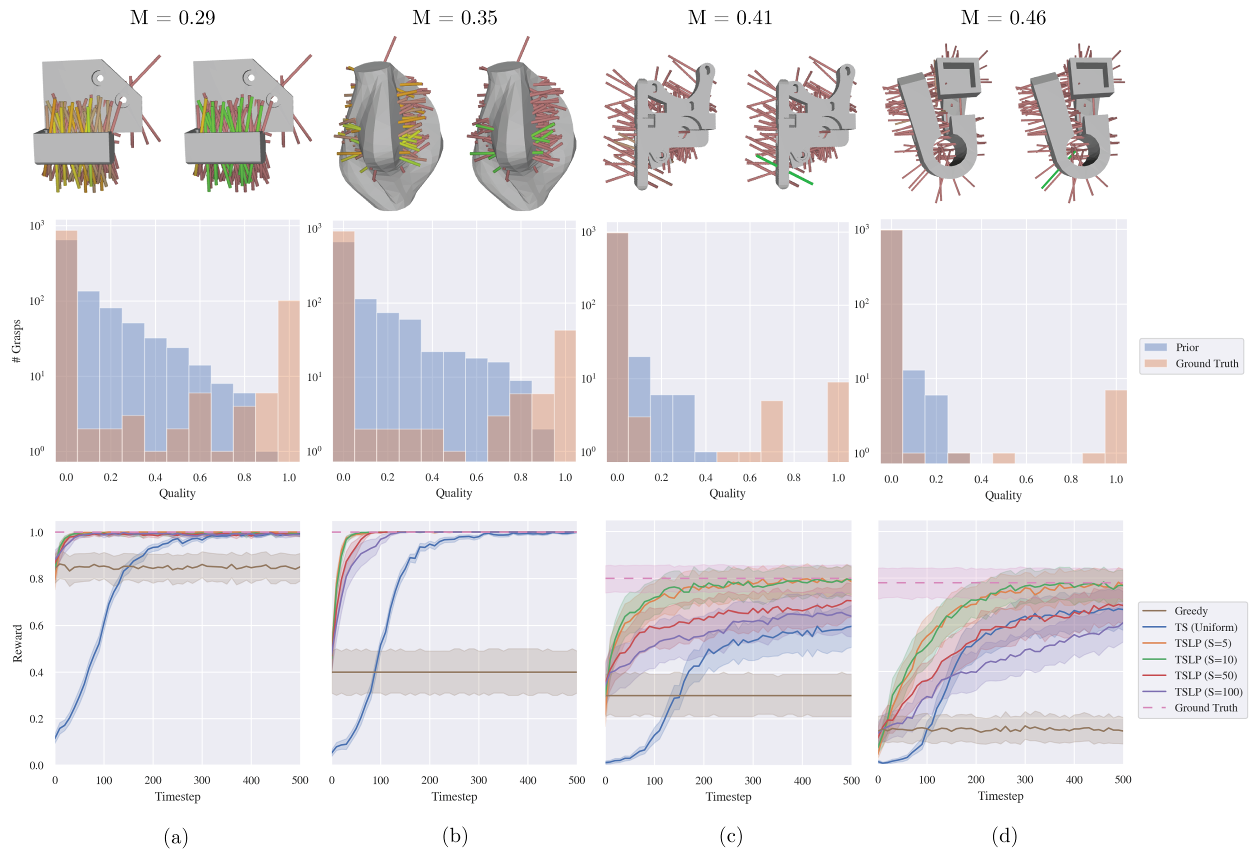


Figure 11.4: Visualization of policy performance for all baselines and TSLP policies (labeled with their prior strength). The first row visualizes grasp qualities as measured by the GQ-CNN prior (left) and the ground truth grasp success probabilities (right) for a single stable pose of each of the four objects (shown top down). Green whiskers indicate high estimated or ground truth grasp quality, while red whiskers indicate low estimated or ground truth grasp quality. In the second row, we visualize the distributions of GQ-CNN prior and ground truth grasp qualities. (a) With a low prior mismatch ($M = 0.29$), the greedy policy performs well and all Thompson sampling policies with non-zero prior strengths converge quickly to the ground truth. (b-c) For objects with higher prior mismatch, the Thompson sampling policies with non-zero prior strength rapidly improve on the prior for object poses with higher prior mismatch ($M = 0.35$ and $M = 0.40$, respectively). (d) For objects with very high prior mismatch ($M = 0.46$), the Thompson sampling policies with non-zero prior strength converge more slowly, but still show improvement on the baseline with prior strength 0.

Table 11.1: **Policy evaluation on large object set:** We evaluate each TSLP policy, the greedy and Thompson sampling with uniform prior baselines, and the ground truth policy on a dataset of 3000 object poses and report the average sum reward over all runs on each of the object poses (300, 000 total training runs per policy, 100 training runs per object for each policy) in the format of mean ± standard deviation. Since we evaluate the policy 51 times per episode, the maximum possible sum reward is 51. For readability, we scale all results by a factor of 100/51 for a maximum scaled sum reward of 100. We find that the est performing TSLP policy, TSLP (S=5), outperforms the greedy baseline by 64.5% while achieving performance within 5.7% of the ground truth oracle baseline.

Greedy	**TS (Uniform)**	**TSLP (S=5)**	**TSLP (S=10)**	**TSLP (S=50)**	**TSLP (S=100)**	**Ground Truth**
54.33 ± 33.02	72.08 ± 20.67	$\mathbf{89.37 \pm 17.88}$	88.43 ± 18.53	82.63 ± 23.51	78.88 ± 26.29	94.53 ± 13.49

sources of uncertainty: (1) the variability in the arms sampled on the initial observation o, and (2) the inherent stochasticity during learning given a set of arms. To control for variations in these parameters, when reporting results on a particular pose of an object, 10 different sets of $K = 100$ arms are sampled on the corresponding observation o. Then, for each of these sets of arms, every policy is trained 10 times for a total of 100 rollouts for each object pose.

We additionally compare the learned policy to a greedy policy that repeatedly selects the grasp with highest quality under GQ-CNN as in [294] and a ground truth oracle policy, which repeatedly selects the grasp with the highest quality under the ground truth grasp quality metrics computed in simulation. The former gives an idea of policy performance if no online exploration is performed, while the latter provides an upper bound on possible performance since it can access the true grasp success probabilities, which are not available to our algorithm.

11.5.2 Simulation Experiments

We conduct simulation experiments across object poses with a wide range of prior mismatches M, as shown in Figure 11.3(a), which plots the frequency of prior mismatch values over the 3946 total object poses in the dataset. When the prior mismatch is relatively low, we expect policies which give more weight to the prior to perform well, while if the prior mismatch is high, we expect policies which prioritize online exploration over following the prior to attain higher rewards.

We evaluate each policy on 3000 of these object poses and compute the sum reward of all policies averaged over the 300, 000 total training runs (100 training runs per object pose). The results are shown in Figure 11.3(b) and Table 11.1. Figure 11.3(b) shows policy performance as a function of prior mismatch, given the distribution of objects over prior mismatch values shown in Figure 11.3(a) over 3000 total object poses. These results suggest that the metric introduced here accurately models prior mismatch, as increased prior mismatch causes performance for all online learning policies, as well as the greedy policy, to degrade. A second trend is that object poses with higher prior mismatch also tend to have lower ground truth quality values, suggesting that GQ-CNN especially struggles to identify high quality grasps when very few are present or when the highest quality grasps have comparatively lower quality.

Table 11.1 shows that TSLP significantly outperforms the greedy baseline and is able to achieve average total reward that is very close to the ground truth policy. This result suggests that TSLP is able to successfully leverage priors from GQ-CNN to outperform GQ-CNN on a wide variety of objects of varying geometries.

As a further case study, we select a set of 4 objects, as shown in Figure 11.4, which are diverse in their shapes and sizes and vary widely in their prior mismatch M. As expected, the ground truth policy (GT) achieves the best performance since it uses oracle information. We find that for objects with relatively low prior mismatch ($M = 0.29$), the greedy policy and the Thompson sampling policies which place very high weight on the GQ-CNN prior (high prior strength) perform very well. However, for objects with higher prior mismatch ($M = 0.35$, $M = 0.41$), we find that the greedy policy performs much more poorly, and online exploration is critical to finding high quality grasps. However, even with high prior mismatch, the gap in performance between the Thompson sampling policies that use the prior and the uniform prior Thompson sampling policy indicates that the GQ-CNN prior helps accelerate grasp exploration substantially. Finally, for objects with very high prior mismatch ($M = 0.46$), the greedy policy and Thompson sampling policies with high prior strengths perform poorly, as expected. However, Thompson sampling policies with low prior strength outperform Thomspon sampling with a uniform prior. This result indicates that although the prior is of very low quality, it still provides useful guidance to the Thompson sampling policy if a low prior strength is used.

Figure 11.5 shows how the mean Bernoulli parameter inferred by TSLP evolves over learning steps for each of the sampled arms. TSLP is able to successfully learn grasp qualities close to the ground truth grasp qualities for a wide variety of different objects. Note that the learned policy is generally more accurate for higher quality grasps, which makes sense since Thompson sampling directs exploration towards high reward grasps, allowing it to focus on distinguishing between high quality grasps rather than capturing the quality distribution of low quality grasps. For the first object, TSLP is able to find the best grasp when the prior strength is relatively weak, but performs poorly when the prior strength is set too high. For the second object, the prior mismatch is lower, so increasing the prior strength accelerates discovery of the best grasps on the object. Note that with a uniform prior, Thompson sampling is generally able to discover most of the best grasps, but fails to distinguish them from bad grasps, resulting in poorer policy performance when these grasps are sampled during policy evaluation.

11.6 Discussion and Conclusion

In this chapter, we present Thompson Sampling with Learned Priors (TSLP), a bandit exploration strategy for robotic grasping which facilitates use of expressive neural network-based prior belief distributions and enables efficient online exploration for objects for which this prior is inaccurate. We quantify the notion of prior mismatch as it pertains to the ranking of arms and explore the effect of prior strength on the efficiency and efficacy of online learning. Experiments suggest that across a dataset of 3000 object poses, TSLP outperforms both a greedy baseline as well as a Thompson sampling baseline that uses a uniform prior and is able to leverage a GQ-CNN prior to significantly accelerate grasp exploration.

11.7 Discussion and Future Work

In future work, we will design new online learning algorithms to explore grasps across different object stable poses and extend experiments to new grasping modalities, such as suction. In addition, we will explore ways to approximately enforce pose consistency in physical experiments. For example, we can use a string to lift the object after each grasp and put it into pose. Additionally, we can detect stable pose changes by evaluating whether the observed depth image changes in a way that cannot be described by a planar rotation and translation. Using the Super4PCS algorithm [314], we can compute the registration of the new point cloud with respect to the original point cloud and restrict the range of output to planar transformations. If the algorithm cannot find such a planar transformation, we resample grasps on the new pose.

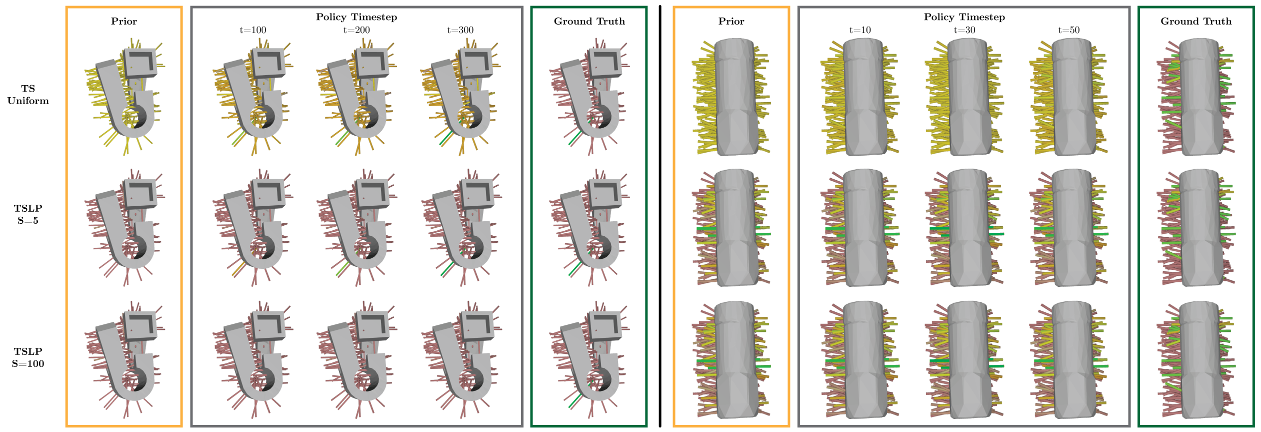


Figure 11.5: We visualize the evolution of the mean Bernoulli parameter (defined in Equation (11.2)) inferred by TSLP with varying prior strengths on sampled arms over learning steps for two different objects. Grasps with high estimated success probabilities or ground truth quality values are colored green, while those with low estimated success probabilities or ground truth qualities are colored orange or red. The inferred mean Bernoulli parameter for TSLP eventually converges to the ground truth probabilities. For the first object, we note that TSLP is able to find the best grasps when the prior strength is relatively weak, but unable to do so when the prior strength is too high since the prior is overly pessimistic ($M = 0.46$). For the second object, the prior is relatively good ($M = 0.31$), so increasing the prior strength accelerates discovery of the best grasps.

Chapter 12

Exploratory Grasping: Asymptotically Optimal Algorithms for Grasping Challenging Polyhedral Objects

Robot grasping systems have a broad array of applications in manufacturing, warehousing, assistive robotics, and household automation [315, 173, 293, 294]. There has been significant recent work in analytic grasp planning algorithms [287, 289, 286, 316], but these methods can often be difficult to apply when object geometry is unknown. In recent years, robot learning has shown exciting promise by utilizing large-scale datasets of previously attempted grasps both in simulation [294, 292, 315, 306] and in physical experiments [297, 293, 315, 317] to learn general-purpose grasping policies that can generalize to objects of varying geometries. To further enable generalization, there has also been work on applying reinforcement learning to learn grasping policies [173, 293, 318, 319]. However, while these techniques have shown significant success in practice, their generality comes at a cost: a policy which aims to grasp every object may fail to generalize to specific challenging objects [290, 320].

Inspired by infants that repeatedly attempt to grasp a toy until they can learn reliable ways to grasp it, we consider a novel problem: Exploratory Grasping, where a robot is presented with an unknown object and learns to reliably grasp it by repeatedly attempting grasps and allowing the object pose to evolve based on grasp outcomes. The objective is for the robot to explore grasps across different object poses to reliably grasp the object from any of its stable resting poses. We then present an algorithm for Exploratory Grasping motivated by polyhedral objects with sparse grasps and adversarial geometries, which can cause persistent failures in general-purpose grasping systems [320, 290]. The intuition is that while a general-purpose grasping policy can be broadly applied to a large set of objects, it can be complemented by learned policies for specific challenging objects.

This chapter contributes (1) Exploratory Grasping: an MDP formulation of the problem of discovering robust grasps across object poses through online interaction that is independent of the grasping modality (parallel-jaw, suction, multi-fingered, etc), (2) parameter-dependent performance bounds on a family of existing tabular reinforcement learning algorithms in this setting, (3) an efficient bandit-inspired algorithm, Bandits for Online Rapid Grasp Exploration Strategy (BORGES), for Exploratory Grasping, and associated no-regret guarantees, (4) simulation experiments suggesting that BORGES can significantly outperform baseline algorithms which explore via reinforcement learning or select actions using general-purpose grasping policies across 46 objects in both the Dex-Net adversarial and EGAD! object datasets, and (5) initial physical results suggesting that BORGES can improve grasp success rate by 45% on average over two challenging objects compared to a Dex-Net baseline.

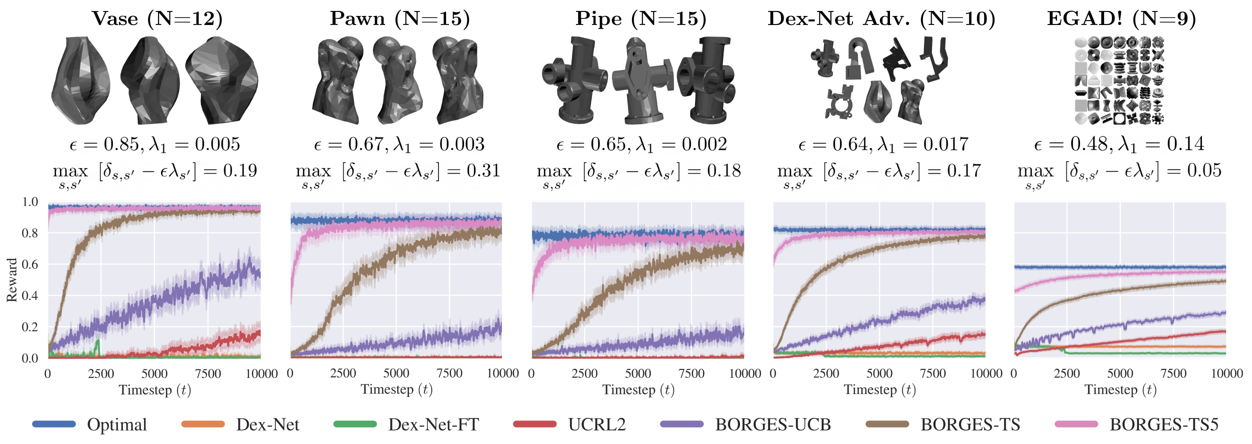


Figure 12.1: **Simulation Experiments:** Performance of each policy across the vase, pawn and pipe objects from the Dex-Net 2.0 adversarial object set [300] (first three columns), as well as aggregated performance over 7 Dex-Net objects (fourth column) and 39 EGAD! objects (fifth column). We report the number of distinguishable stable poses (N) with the first three shown in the second row, as well as ϵ (ground truth value for the lowest quality best grasp across poses), λ_1 (least likely stable pose probability) and $\max_{s,s'} [\delta_{s,s'} - \epsilon\lambda_{s'}]$ (maximum difference in transitioning via toppling and transitioning via grasping to a new pose) values. For the datasets, we give mean values and show the most likely stable pose of each object. We visualize policy performance in the learning curves in the lowest row. BORGES quickly converges to near-optimal performance even when Assumption 12.4.1 is violated while the other algorithms fail to reach optimal performance.

12.1 Related Work

Work in analytic robot grasping assumes knowledge of object geometry and pose to design geometric grasp planning algorithms Bicchi and Kumar [287], Rimon and Burdick [286], Kim et al. [316], and Murray [289]. Recently, learning-based general-purpose algorithms have emerged for planning grasps robust to uncertainty in object geometry and sensing on a wide range of objects with data-driven strategies Mahler et al. [294], Pinto and Gupta [297], Viereck et al. [306], Morrison, Corke, and Leitner [315], Lenz, Lee, and Saxena [291], and Kappler, Bohg, and Schaal [292] and online exploration through reinforcement learning Kalashnikov et al. [173] and Levine et al. [293]. The latter approaches have been very effective in learning end-to-end policies for grasp planning for a variety of objects. In contrast, we focus on learning policies for the specific challenging objects that pose problems for these systems Wang et al. [290] and Sanders et al. [320].

Significant prior work studies multi-armed bandit frameworks for online grasp exploration Laskey et al. [321, 298], Oberlin and Tellex [322], Li et al. [323], Kroemer et al. [324], Eppner and Brock [325], and Lu, Van der Merwe, and Hermans [326], but primarily focus on settings when some geometric knowledge is known or grasp exploration is limited to a single object pose. In contrast, we consider a formulation where the robot must learn grasps across all poses of the object without human supervision. Laskey et al. [298] consider the setting where some prior geometric knowledge is known, but present an algorithm for 2D objects that does not use visual inputs. Li et al. [323] and Oberlin and Tellex [322] relax these assumptions by exploring grasps for a single stable pose of 3D object with RGB or depth observations. We extend these ideas by repeatedly dropping the object and exploring grasps in all encountered stable poses. Thus, BORGES naturally explores grasps over the distribution of possible stable poses Goldberg et al. [327] to learn a robust policy which can reliably grasp the object when randomly dropped in the workspace.

A key requirement for successful exploration of grasps on an object is exploring the space of its resting stable poses, since the object will necessarily be in one of these poses during grasp attempts. There has been significant prior work on orienting parts into specific stable poses through a series of parallel jaw gripper movements Goldberg [328], toppling actions Correa et al. [329], and squeezing actions Goldberg and Mason [330]. However, these approaches require knowledge of object geometry apriori. When object geometry is not known, but assumed to be polyhedral, prior work Goldberg et al. [327] has established that repeatedly dropping the object from a known initial distribution of poses onto a flat workspace results in a stationary distribution over stable poses. This dropping procedure provides a useful primitive for reaching new stable poses of an object. We leverage this insight to formulate the Exploratory Grasping problem and design algorithms that address this setting to discover high-quality grasps for different object stable poses.

12.2 Exploratory Grasping: Problem Statement

Given a single unknown polyhedral object on a planar workspace, the objective is to learn a grasping policy that maximizes the likelihood of grasp success over all stable poses of the object Goldberg et al. [327] and Moll and Erdmann [331]. We formulate Exploratory Grasping as a Markov Decision Process (Section 12.2.1), define assumptions on the environment

(Section 12.2.2) and formulate the policy learning objective (Section 12.2.3).

12.2.1 Exploratory Grasping as an MDP

We consider exploring grasps on an unknown, rigid, polyhedral object $\mathcal{O}$ which rests in one of a finite set of N distinguishable stable poses with associated landing probabilities $\{\lambda_s\}_{s=1}^N$. We consider polyhedral objects since they admit a finite number of stable resting poses, as does any object defined with a triangular mesh Rao and Goldberg [332]. We assume that the robot does not initially know any of these stable poses or their count N. The robot must discover new stable poses from experience by attempting grasps on the object in its current stable pose and lifting and then releasing the object when grasps are successful. We assume an overhead depth camera with known camera intrinsics that cannot reliably determine the 3D shape of the object, but can be used to recognize distinguishable 3D poses by performing planar translations and rotations of the image into a canonical orientation and translation. We also assume that the camera can be used after grasp attempts to determine if a grasp succeeds.

We model Exploratory Grasping as an MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R)$ as follows. We first define a one-to-many mapping from the set of object stable poses Σ to the set of overhead point clouds $\mathcal{I}$ that are scale-invariant, translation-invariant, and rotationally-invariant about the vertical axis. Then, we define the state space as the set of *distinguishable* stable poses $\mathcal{S}$, where $\mathcal{S}$ is the set of equivalence classes within Σ, where two poses are equivalent if they map to the same set of overhead point clouds $\mathcal{I}$. For example, all poses of a cube would map to the same overhead point cloud, so a cube only has 1 distinguishable stable pose. We assume point cloud $\mathcal{I}$ is obtained by deprojecting a depth image observation $o \in \mathbb{R}^+_{H\times W}$ taken from the overhead camera with known intrinsics. We define a set $\mathcal{A}_s$ of K grasp actions, such as parallel-jaw or suction grasps, for each stable pose $s \in \mathcal{S}$ of the object. The K grasps are sampled from o for each stable pose as in Mahler et al. [300]. The full action space is the union of the grasp actions available on each pose: $\mathcal{A} = \bigcup\limits_{s\in\mathcal{S}} \mathcal{A}_s$. We assume that the robot acts in an environment with unknown transition probability distribution $P(s' \mid s, a)$, which denotes the probability of the object transitioning to pose s' if grasp action a is executed in pose s. If a is a successful grasp, then the object is released onto the workspace to sample a new stable pose s' based on unknown landing probabilities $\{\lambda_s\}_{s=1}^N$, while if a is a failed grasp, the object topples into some new pose s' with unknown probability $\delta_{s,s'}$. The robot receives reward $R(s, a)$ for selecting grasp action a on stable pose s, so $R(s, a) = 1$ if executing a in stable pose s results in the object being successfully grasped and lifted, and 0 otherwise. We assume that rewards are drawn from a Bernoulli distribution with unknown parameter $\phi_{s,a}$: $R(s, a) \sim \text{Ber}(\phi_{s,a})$ for $a \in \mathcal{A}_s$.

12.2.2 Assumptions

To study Exploratory Grasping, we first establish assumptions on the system dynamics to ensure that all poses are reachable and which describe how the object pose can evolve when the object is (1) released onto the workspace or (2) when a grasp is attempted. We make no assumptions on the grasping modality (e.g., parallel jaw, suction, multi-fingered, etc).

Assumption 12.2.1. ***Grasp Dynamics:*** *If a grasp succeeds, we assume that the robot can randomize the pose of the object before releasing it to sample subsequent stable poses from the associated unknown stable pose landing probabilities* $\{\lambda_s\}_{s=1}^N$ *for* $\mathcal{O}$*. If a grasp fails, we assume that the object's pose will either remain unchanged or topple into some other pose* s' *with unknown probability* $\delta_{s,s'}$*.*

Assumption 12.2.2. ***Release Dynamics:*** *The categorical distribution over stable poses defined by landing probabilities* $\{\lambda_s\}_{s=1}^N$ *is stationary and independent of prior actions and poses when* $\mathcal{O}$ *is released from a fixed height with its orientation randomized as in Goldberg et al. [327].*

Assumption 12.2.3. ***Irreducibility:*** *We assume that there exists a policy* π *such that the Markov chain over stable poses induced by executing* π *in* $\mathcal{M}$ *is irreducible, and thus can reach all stable poses with nonzero probability for any initialization.*

Note that Assumption 12.2.3 is satisfied if, for all poses $s \in \mathcal{S}$, $\lambda_s > 0$ and there exists a grasp with success probability $\epsilon > 0$. We assume that these conditions hold for analysis, but since object toppling is also possible, note that these conditions are sufficient but not necessary for ensuring irreducibility.

12.2.3 Learning Objective

The objective is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected average reward over an infinite time horizon under the state distribution induced by π. Let $\tau = \{(s_t, \pi(s_t))\}_{t=1}^T$ be a trajectory of all states and actions when executing π for time T and let $r(\tau) = \sum_{t=1}^T R(s_t, \pi(s_t))$ be the sum of rewards for all states and actions in τ, and let $p(\tau|\pi)$ be the trajectory distribution induced by policy π. Then the expected average reward obtained from policy π in $\mathcal{M}$ is given as:

$$J(\mathcal{M}, \pi, T) = \frac{1}{T}\mathbb{E}_{\tau\sim p(\tau|\pi)}\left[r(\tau)\right] \tag{12.1}$$

The objective is to find the policy which maximizes expected reward over an infinite time horizon:

$$\pi^* = \arg\max_{\pi} \lim_{T\to\infty} J(\mathcal{M}, \pi, T) \tag{12.2}$$

12.3 Reinforcement Learning for Exploratory Grasping

We first consider the performance of reinforcement learning algorithms for Exploratory Grasping and leverage the structure of the MDP described in Section 12.2 to establish a bound on the cumulative regret for a family of existing tabular reinforcement learning algorithms when applied to Exploratory Grasping. See Section J.1 of the supplementary material for all proofs.

12.3.1 Analyzing the Exploratory Grasping MDP

A common metric to measure policy performance in online-learning settings is *regret*, which has been analyzed in the reinforcement learning setting by a variety of prior work [333, 334, 335, 336]. Intuitively, regret quantifies the difference in accumulated reward within T timesteps between a given policy π and optimal infinite horizon policy π^* for MDP $\mathcal{M}$. More precisely, we define average regret based on the definition in [333]:

$$\text{Regret}(\mathcal{M}, \pi, T) = \max_{\pi'} \left[\lim_{T\to\infty} J(\mathcal{M}, \pi', T)\right] - J(\mathcal{M}, \pi, T) \tag{12.3}$$

Recent theoretical work by Jaksh, Ortner, and Auer [333] on reinforcement learning for tabular MDPs yielded algorithms which can attain average regret proportional to the *diameter* of the MDP, a measure of the furthest distance between pairs of states under an appropriate policy. However, for general MDPs, this diameter can be arbitrarily large, making these regret bounds difficult to interpret in practical settings. We leverage the structure of the MDP to derive an upper bound on the MDP diameter, which precisely quantifies the difficulty of grasp exploration based on the parameters of $\mathcal{M}$.

We begin by defining the Markov chain over $\mathcal{S}$ induced by a stationary deterministic policy π.

Definition 12.3.1. ***Pose Evolution under*** π***:*** *Given stationary policy π, the transitions between pairs of states in $\mathcal{M}$ is defined by a Markov chain. Precisely, the transition probabilities under π, denoted by P^π where $P^\pi[s, s'] = P(s' \mid s, a = \pi(s))$, are given as follows:*

$$P^\pi[s, s'] = \phi_{s,\pi(s)}\lambda_{s'} + (1 - \phi_{s,\pi(s)})\delta_{s,s'} \tag{12.4}$$

Given this Markov Chain over poses for π, we can now analyze the diameter of the MDP, denoted $D(\mathcal{M})$, by considering the hitting time between stable poses in $\mathcal{M}$ as defined in [337].

Definition 12.3.2. *Let $T^\pi_{s\to s'}$ denote the expected hitting time between states s and s' under policy π under the Markov chain defined in Definition 12.3.1. Then the diameter of $\mathcal{M}$ is* *defined as follows [333]:*

$$D(\mathcal{M}) = \max_{s \neq s'} \min_{\pi} \mathbb{E}\left[T^{\pi}_{s \to s'}\right] \tag{12.5}$$

Intuitively, $D(\mathcal{M})$ measures the temporal distance between the most distant states in an MDP under the policy that minimizes this distance. We now leverage the structure of $\mathcal{M}$ to establish an upper bound on $D(\mathcal{M})$.

Lemma 12.3.1. *The diameter of $\mathcal{M}$ can be bounded above as follows:*

$$D(\mathcal{M}) \leq \frac{1}{\epsilon \lambda_1} \tag{12.6}$$

Here ϵ is a lower bound on the success probability of the highest quality grasp over all stable poses and λ_1 is the landing probability for the least likely stable pose.

Lemma 12.3.1 captures the intuition that the diameter of the MDP is large if the best grasp in each stable pose has a low success probability (ϵ is small), or if there exists a stable pose with very low landing probability (λ_1 is small).

Now we can establish regret bounds for 4 tabular reinforcement learning algorithms (UCRL2 [333], KL-UCRL [335], PSRL [338], and UCRLV [334]) when applied to $\mathcal{M}$ by combining diameter dependent regret bounds from prior work and the bound on the diameter of $\mathcal{M}$ established in Lemma 12.3.1.

Theorem 12.3.1. *UCRL2 [333], KL-UCRL [335] and PSRL [338] achieve average regret given by Regret$(\mathcal{M}, \pi, T) \sim \tilde{O}\left(\frac{N}{\epsilon\lambda_1}\sqrt{\frac{K}{T}}\right)$ for any Exploratory Grasping MDP $\mathcal{M}$ while UCRLV [334] achieves average regret given by Regret$(\mathcal{M}, \pi, T) \sim \tilde{O}\left(\sqrt{\frac{NK}{\epsilon\lambda_1 T}}\right)$. Here N, K, T, ϵ and λ_1 are defined as in Section 12.2.*

Theorem 12.3.1 leverages the specific structure of the MDP in Exploratory Grasping to relate the accumulated regret of 4 tabular RL algorithms to intuitive parameters of the MDP, providing insight into the theoretical complexity of Exploratory Grasping in the context of reinforcement learning.

12.4 Bandits for Online Rapid Grasp Exploration Strategy (BORGES)

While the reinforcement learning algorithms in Section 12.3 provide a method and formal guarantees for Exploratory Grasping, the algorithms themselves do not exploit any specific structure in MDP $\mathcal{M}$. One key feature of Exploratory Grasping is that in practice, objects have a small, finite set of stable poses [327]. This motivates learning a set of N policies, each of which explore grasps in a particular object stable pose. However, the grasp exploration problem in each pose is coupled. For example, there may exist a pose s with no available high quality grasps but a high likelihood of a failed grasp causing the object to topple into another pose with high quality grasps. Then, the optimal policy may deliberately fail to grasp the object in pose s in order to obtain access to grasps in the more favorable stable pose, leading it to avoid grasp exploration in poses without high quality grasps. To avoid this behavior, we introduce Assumption 12.4.1.

Assumption 12.4.1. *We assume that $\delta_{s,s'} \leq \epsilon\lambda_{s'}$ for all $s \neq s'$ where $\delta_{s,s'}$ is the probability of toppling into pose s' given a failed grasp in pose s, ϵ is a lower bound on the success probability of the highest quality grasp over all stable poses as defined above, and $\lambda_{s'}$ is the landing probability of pose s'.*

Assumption 12.4.1 ensures that there exists a grasp in all stable poses s such that the probability of transitioning to new pose s' via a grasp attempt is higher than that of toppling from pose s to pose s'. Given this assumption, the optimal grasp exploration policy in $\mathcal{M}$ reduces to selecting the grasp with highest success probability in each encountered pose, as this policy maximizes both reward at the current timestep and exploration of other stable poses when the object is released. In other words, the global optimal policy is the *greedy* policy. Given this structure, we can view the grasp exploration problem as N independent multi-armed bandit problems corresponding to grasp exploration in each pose. However, although grasp exploration can be performed independently in each pose, the success of a grasp exploration policy in one pose affects the time available to explore grasps in another pose, thereby coupling each bandit problem.

We propose an algorithm that takes advantage of this structure to enable rapid online grasp exploration: Bandits for Online Rapid Grasp Exploration Strategy (BORGES)[1]. BORGES works by maintaining the parameters of N independent bandit policies $(\pi_s^{\mathcal{B}})_{s=1}^N$, where $\pi_s^{\mathcal{B}} : s \rightarrow \mathcal{A}_s$ and $\pi_s^{\mathcal{B}}$ is only *active* in pose s. Let $\pi^{\mathcal{B}}$ denote the meta-policy induced by executing $\pi_s^{\mathcal{B}}$ in pose s and assume that $\pi_s^{\mathcal{B}}$ is learned by running a no-regret online learning algorithm $\mathcal{B}$ for grasp exploration in s. No-regret algorithms for the stochastic multi-armed bandit problem include the UCB-1 algorithm [339] and Thompson Sampling [340]. We then formulate a new notion of regret capturing the gap between $\pi^{\mathcal{B}}$ and the optimal policy on their respective distributions and show that BORGES achieves vanishing average regret despite the interdependence between pose exploration times.

Let $p_T^{\mathcal{B}}$ denote the distribution of poses seen under the sequence of policies $\pi_{1:T}^{\mathcal{B}}$ at each round of learning up to time T and let p_T^* denote the distribution of poses seen when executing the optimal policy (π^*) in $\mathcal{M}$ up to time T. We define the average regret achieved by BORGES in MDP $\mathcal{M}$ after T rounds as the difference in accumulated reward of the optimal policy on pose distribution p_T^* and the accumulated reward of $\pi^{\mathcal{B}}$ on pose distribution $p_T^{\mathcal{B}}$.

Definition 12.4.1. *The average regret accumulated by running $\mathcal{B}$ in each stable pose is defined as the difference between the average regret on each pose visited by the optimal policy*

[1]Jorge Luis Borges (1899-1986) was a brilliant writer whose short stories considered geometry, time, and combinatorics.

π^* weighted by the probability of it visiting each pose and the corresponding quantity for the executed policy $\pi^{\mathcal{B}}$:

$$\mathbb{E}\left[\mathcal{R}^{\mathcal{B}}(T)\right] = \sum_{s=1}^{N} p_T^*(s)\mathbb{E}\left[\frac{1}{T_s^*}\sum_{t=1}^{T_s^*} R(s, \pi^*(s))\right] - \sum_{s=1}^{N} p_T^{\mathcal{B}}(s)\mathbb{E}\left[\frac{1}{T_s^{\mathcal{B}}}\sum_{t=1}^{T_s^{\mathcal{B}}} R(s, \pi_t^{\mathcal{B}}(s))\right]$$

where T_s^* is the time spent by the optimal policy in pose s and $T_s^{\mathcal{B}}$ is the time spent by $\pi^{\mathcal{B}}$ in pose s.

In Section J.1 of the supplementary material, we show that the average regret as defined in Definition 12.4.1 vanishes to 0 in the limit as $T \to \infty$ as stated in Theorem 12.4.1.

Theorem 12.4.1. *The average regret achieved by BORGES, when using any no-regret bandit algorithm $\mathcal{B}$ for grasp exploration in each encountered stable pose, vanishes in the limit:*

$$\lim_{T\to\infty} \mathbb{E}\left[\mathcal{R}^{\mathcal{B}}(T)\right] = 0$$

This result leverages the precise structure of $\mathcal{M}$, namely that the optimal policy is the greedy policy, to provide sublinear regret guarantees for BORGES.

A convenient property of BORGES is that while exploring grasps on an object, it naturally explores different object poses as well. Theorem 12.4.2 establishes an upper bound on the expected number of timesteps required for BORGES to reach all object stable poses.

Theorem 12.4.2. *Let T_{cover} denote the number of grasps BORGES executes until it has reached every stable pose. We can bound $\mathbb{E}[T_{cover}]$ above as follows where K, ϵ, and λ are defined as in Section 12.2.*

$$\mathbb{E}\left[T_{cover}\right] \leq \frac{K}{\epsilon}\sum_{j=1}^{N}\frac{1}{1-\sum_{i=2}^{j}\lambda_{i-1}}$$

We note that there can be pathological objects for which Assumption 12.2.3 is violated due to a sink state from which the object can neither be grasped nor toppled into a new pose. In this case, no grasp exploration strategy will be successful, because there will exist poses from which the robot cannot recover. However, one benefit of BORGES is that it can detect these cases, as it maintains estimates of the success probabilities of each grasp during learning and thus could request external intervention from a human supervisor when it becomes clear that a given pose is a sink state in $\mathcal{M}$. We also acknowledge that many objects violate Assumption 12.4.1 as they can be easily toppled between poses, including several of the objects we use for evaluation. However, we find that even when Assumption 12.4.1 is violated, BORGES still achieves good performance in practice (Figure 12.1).

12.5 Experiments

12.5.1 Simulation Experiments

In simulation experiments, we study whether BORGES can discover high quality grasps on objects for which general-purpose grasping policies, such as Dex-Net 4.0 [294] and GG-CNN Morrison, Corke, and Leitner [315], perform poorly and whether learning different policies for each stable pose accelerates grasp exploration.

Simulation Details: To evaluate each policy, we choose 7 objects from the set of Dex-Net 2.0 adversarial objects [300] and 39 evaluation objects from the EGAD! dataset Morrison, Corke, and Leitner [341] for which general-purpose parallel-jaw grasping policies (Dex-Net 4.0 [294] and GG-CNN [315]) perform poorly, despite there existing high-quality grasps in multiple stable poses. We sample a set of K parallel-jaw grasps on the image observation of pose s of each object as in [300], and calculate the ground-truth quality of each grasp, $\phi_{s,a}$, using a wrench resistance metric that measures the ability of the grasp to resist gravity [308]. We randomize the initial pose of each object and execute BORGES and baselines, sampling rewards from $\text{Ber}(\phi_{s,a})$. If the grasp succeeds, we randomize the pose, choosing a stable pose according to the stable pose distribution of the object. Otherwise, the object may topple into a new stable pose. We execute each policy for 10 rollouts of 10 trials, where new grasps are sampled for each trial and each rollout evaluates the policy over 10,000 timesteps of grasp exploration. Since there is stochasticity in both the grasp sampling and the policies themselves, we average policy performance across the 10 rollouts and 10 trials. In addition, we smooth policy performance across a sliding window of 20 timesteps and report average reward for each timestep.

To model object toppling when grasps are unsuccessful (Section 12.2.2), we use the toppling analysis from Correa et al. [329] to determine the toppling transition matrix for a given object. Specifically, we generate the distribution of next states from a given state by sampling non-colliding pushes across vertices on the object, finding their distribution of next states given perturbations around the nominal push point, and average the distribution from all of the pushes. Then if a grasp fails, we choose the next state according to the corresponding topple transition probabilities. When evaluating policies, we remove poses that have no grasps with nonzero ground-truth quality and renormalize the stable pose distribution. We emphasize that we perform this step for all policies and do this only to ensure that no poses act as sink states from which the robot can never change the pose in the future (and thus ensure that Assumption 12.2.3 is satisfied). For the 7 objects from the Dex-Net 2.0 adversarial object dataset, 11% of poses (an average of 1.3 poses per object) were removed. We present additional experiments in the supplementary material where this pose removal step is not performed, and find that while all policies perform more poorly (Section J.3), BORGES still outperforms all baseline policies. We also include further details on the experimental setup in the supplementary material.

Policies: We compare 3 variants of BORGES against 3 baselines to evaluate whether BORGES is able to (1) substantially outperform general-purpose grasping policies Mahler et al. [294] on challenging objects and (2) learn more efficiently than other online learning algorithms which update a grasp quality convolutional network (GQCNN) online or explore grasps via reinforcement learning. We also instantiate BORGES with different algorithms for $\mathcal{B}$ to study how this choice affects grasp exploration efficiency and implement an oracle baseline to study whether BORGES is able to converge to the optimal policy. We compare the following baselines and BORGES variants: **Dex-Net**, which selects grasps greedily with respect to the Grasp Quality Convolutional Network from Dex-Net 4.0 [294] with probability 0.95 and selects a grasp uniformly at random with probability 0.05, **Dex-Net-FT**, which additionally fine tunes the GQCNN policy online from agent experience, an implementation of **UCRL2** from [342], a tabular RL algorithm discussed in Section 12.3, and instantiations of BORGES with the UCB-1 algorithm [339] (**BORGES-UCB**), Thompson sampling (**BORGES-TS**) with a uniform Beta distribution prior, and Thompson sampling with a prior from the Dex-Net policy of strength 5 (**BORGES-TS5**) as in [323]. Finally, we implement an oracle baseline that chooses grasps with the best ground-truth metric at each timestep to establish an upper bound on performance.

Simulation Results: Results for baselines and BORGES variants are in Figure 12.1. Above the learning curves in Figure 12.1, we report the maximum violation of Assumption 12.4.1 across all pose pairs for each object ($\max_{s,s'}[\delta_{s,s'} - \epsilon\lambda_{s'}]$). This value captures how much more likely an object is to topple between poses s and s' than be successfully grasped in s and released into s' for pose pair (s, s') which most violates Assumption 12.4.1. BORGES is robust to large violations of Assumption 12.4.1, and substantially outperforms prior methods even when there exist pose pairs where it is 30% more likely to topple between the poses than transition between them with a successful grasp.

As shown in Figure 12.1, the Dex-Net policy typically performs very poorly, achieving an average reward of less than 0.1 per timestep. While the online learning policies also start poorly, they quickly improve, and BORGES-TS eventually converges to the optimal policy. We also find that BORGES-TS5, which leverages Dex-Net as a prior using the method presented in [323], further accelerates convergence to the optimal policy. This result is promising, as it suggests that the exploration strategy in BORGES can be flexibly combined with general-purpose grasping policies to significantly accelerate grasp exploration on unknown objects. We find that the Dex-Net-FT policy performs very poorly even though it continues to update the weights of the network online with the results of each grasp attempt and samples a random grasp with probability 0.05, which aids exploration. We hypothesize that this is due to the initial poor performance of Dex-Net—since the vast majority of fine-tuning grasps attain zero reward, the network is unable to explore enough high-quality grasps on the object. Overall, these results suggest that BORGES can greatly increase grasp success rates on objects for which Dex-Net performs poorly. We also find that BORGES policies greatly outperform tabular RL policy UCRL2 by leveraging the structure of the MDP. Both the UCB and Thompson sampling implementations maintain separate policies in each pose, leveraging the fact that the optimal policy is the greedy policy. This allows the BORGES policies to not waste timesteps exploring possible transitions to poses with low rewards. Thompson sampling additionally leverages the fact that the rewards are distributed

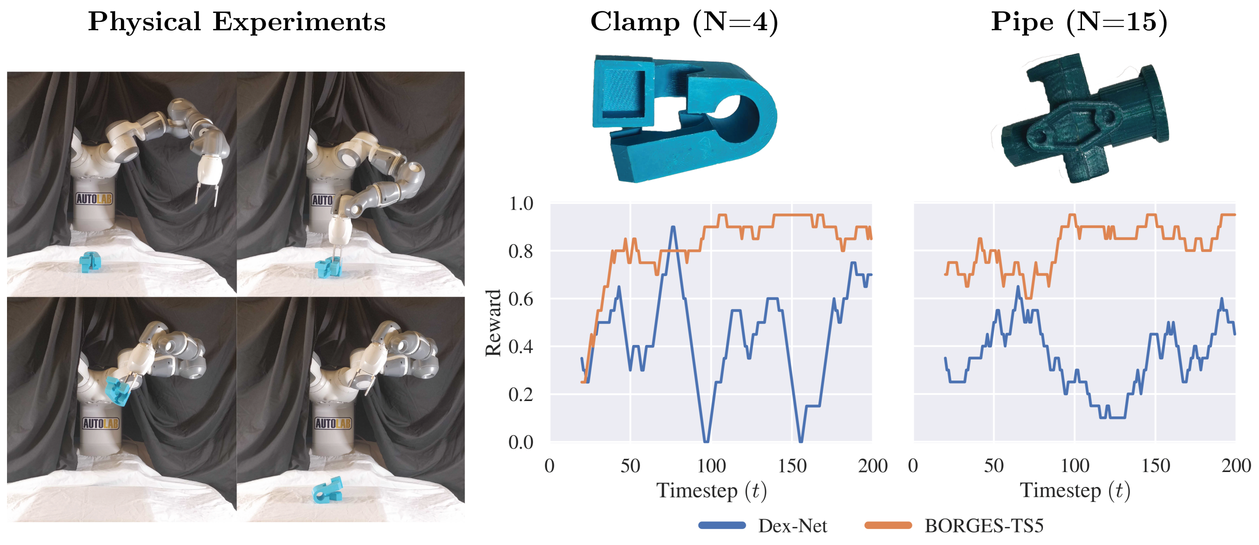


Figure 12.2: Experiment setup and learning curves for the Dex-Net and BORGES-TS5 policies for two objects across 200 grasp attempts (smoothed with a running average of 20 attempts). The robot attempts to grasp the object at each timestep and, if it succeeds, rotates and drops the object to sample from the stable pose distribution (left). BORGES-TS5 quickly converges within 100 attempts on both objects, indicating that it finds grasps that succeed nearly every time for each pose. Dex-Net's performance remains uneven, indicating that it finds high-quality grasps for some poses, but not others.

as Bernoulli random variables, which may explain the significant performance gap between the Thompson sampling and UCB implementations. Thus, BORGES policies quickly learn to choose high-quality grasps in each pose to transition quickly to new poses.

We perform sensitivity analysis of BORGES to ϵ and λ_1 and find that BORGES quickly converges to the optimal policy unless ϵ or λ_1 is low. We additionally evaluate BORGES for different numbers of sampled grasps (K) on each pose, and find that with decreasing values of K, all policies perform worse since the likelihood of sampling a high quality grasp decreases, but BORGES policies continue to outperform prior methods. See Section J.4 of the supplementary material for more details.

12.5.2 Initial Physical Experiments

In physical experiments, we evaluate BORGES on two challenging objects on an ABB YuMi robot and compare performance with the Dex-Net policy from Section 12.5.1. Section J.5 of the supplementary material contains further details on the experimental setup and procedures. Figure 12.2 shows learning curves for both BORGES and Dex-Net; results suggest that BORGES quickly finds high-quality grasps across all object poses, converging to near-perfect performance. Dex-Net correctly predicts high-quality grasps on some of the poses, but attempts suboptimal grasps in others, resulting in high variance performance across the rollout since it is unable to adapt to its successes and failures. Across the final 100 timesteps for each object (i.e., after BORGES begins to explore all poses), BORGES reaches 0.89 and 0.87 success rates on the clamp and pipe, respectively, as compared to 0.49 and 0.37 for the Dex-Net policy after just 200 grasp attempts in real world. This highlights the importance of a policy that can learn online from successful and failed grasp attempts; Dex-Net does not learn online so continuing to attempt grasps will lead to the same high-variance behavior over time, but BORGES continues to stabilize as it approaches optimal performance across all stable poses.

12.6 Future Work

In future work, we plan to evaluate BORGES in extensive physical trials with different grasping modalities such as suction grasps and with priors from different grasp planners. One property of BORGES is that in the process of exploring grasps, it also explores different object stable poses. Thus, we are excited to explore further applications of BORGES; for example, exploring different object poses to construct accurate 3D models of unknown objects or inspect parts for defects.

Part V

Conclusion and Future Work

Chapter 13

Conclusion

This dissertation contributes a number of online learning algorithms which enable robotic agents to learn new tasks by exploring the environment while ensuring that this exploration is both efficient and safe enough to be practical on robotic systems in the real world. To do this, we made the observation that while online learning is critical for robotic systems to be able to continually improve in their performance and align their control strategies with the properties of the environment, online learning algorithms which learn purely from autonomous execution are rarely practical in robotic settings due to inefficiency and safety concerns, which are of extreme importance when deploying control policies on physical hardware. This motivates developing methods to scalably supervise these algorithms that can convey additional information about promising and unpromising behaviors without forcing the robot to identify all of these behaviors through online exploration. To this end, we presented algorithms which leverage offline datasets or sparing human queries to structure online exploration and demonstration how this enables safe and efficient online robot learning in the physical world.

In Chapters 2-3, we presented LazyDAgger and ThriftyDAgger, interactive imitation learning algorithms which can adaptively request human intervention in order to improve policy performance while limiting burden on the human supervisor. Results suggest that LazyDAgger and ThriftyDAgger are able to more effectively balance task performance and supervisor burden than prior interactive imitation learning algorithms. Notably, ThriftyDAgger often achieves superior task performance than a baseline in which a human constantly monitors the robot and decides when to intervene, suggesting that ThriftyDAgger can sometimes identify opportune times for human intervention more effectively than a human themselves. Then, in Chapter 4, we showed how ideas from online imitation learning and model-based reinforcement learning can be combined to design an algorithm which can achieve the high sample efficiency of model-based reinforcement learning algorithms while maintaining the fast policy evaluation enabled by model-free policies.

Then in Chapters 5-8, we studied how a small set of suboptimal demonstrations can be used to substantially accelerate exploration in reinforcement learning. We explored two classes of approaches: (1) learning a safe set to constrain exploration to the neighborhood of prior successful trajectories (Chapters 5-7) and (2) using demonstrations to define a modified Bellman backup to boost the Q-values of transitions in demonstration trajectories (Chapter 8). We found that these algorithms enable reinforcement learning for long horizon, sparse reward tasks while outperforming the suboptimal demonstrations. In Chapters 9-10, we studied how negative demonstrations, which contain examples of safety violations, can be used to design reinforcement learning algorithms which can avoid unsafe states during exploration.

We ended by exploring bandit-based exploration algorithms for efficient robot grasping in the real world. In Chapter 11, we presented an algorithm to learn priors on grasp quality from general purpose grasping systems such as Dex-Net [300] and show how these priors can significantly accelerate grasp exploration on a given object stable pose in simulation. In Chapter 12, we extended these ideas to rapidly exploring grasps on multiple different object stable poses both in simulation and on a physical robot. We found that this synthesis of priors from general purpose grasping systems trained on large diverse datasets of objects and efficient online exploration algorithms from the multi-armed bandit literature makes it possible to simultaneously leverage the generalization ability of neural networks trained on large datasets and the rapid adaptability of classical online learning methods.

13.1 Opportunities for Future Work

There are a number of exciting areas to explore in future work related to each of the main sections of this dissertation. We discuss future work related to efficient online imitation learning algorithms in Section 13.1.1, reinforcement learning from suboptimal demonstrations in Section 13.1.2, reinforcement learning from negative demonstrations in Section 13.1.3, and bandit-based grasp exploration in Section 13.1.4.

13.1.1 Efficient Online Imitation Learning

In the context of interactive imitation learning, it would be interesting to study how recent work on estimating uncertainty in neural networks [343] could be applied to improving the intervention criteria of LazyDAgger and ThriftyDAgger. It would also be interesting to precisely study the settings in which human and learned intervention criteria disagree, or use the points at which humans chose to initiate interventions previously to learn when a human might choose to intervene in the future. Studying application of interactive imitation learning algorithms to large scale robot fleet learning problems in autonomous driving or warehouse automation would also be exciting to explore in the future.

13.1.2 Reinforcement Learning from Suboptimal Demonstrations

There are also a variety of exciting extensions for the algorithms we discussed on reinforcement learning from suboptimal demonstrations in Chapters 5-8. One interesting direction could be to extend Adjustable Boundary Condition LMPC (ABC-LMPC) in Chapter 5 to high-dimensional settings as studied in Chapter 7. It would also be interesting to learn goal-conditioned safe sets and extend the ideas from ABC-LMPC, SAVED, and LS^3 to design motion planning algorithms for stochastic dynamical systems by using the strategy in ABC-LMPC to iteratively expand their controller domain throughout the state space in order to find a sequence of controllers to robustly plan between any pair of states. It would also be exciting to explore recent work in support and density estimation [344] to improve the quality of learned safe sets in high-dimensional state spaces.

13.1.3 Reinforcement Learning from Negative Demonstrations

It would also be interesting to study extensions of the constrained reinforcement learning algorithms in Chapters 9-10. One direction is to study how to collect informative offline datasets with constraint violations which would inform safely learning a variety of different downstream tasks in the environment while not violating constraints more often than is strictly necessary for learning safely after offline data collection. It would also be interesting to use learned safety critics to determine when human intervention might be necessary rather than just for querying/learning a recovery policy.

13.1.4 Learning Priors for Rapid Bandit-Based Grasp Exploration

Finally, for the grasping projects in Chapters 11-12, it would be exciting to study how recent work from the meta-learning community [345] might be applied to learn initializations for grasp quality prediction networks that can be rapidly fine-tuned on new objects. This could be used in conjunction with bandit based exploration to further accelerate grasp exploration in practice. It would also be exciting to learn priors which quantify some notion of uncertainty in their predictions, as this may be useful in deciding how much to trust these priors when choosing grasps during online exploration.

13.2 Broader Vision for Robot Learning

I am generally interested in studying scalable ways to supervise online learning algorithms for robot learning applications. The two primary bottlenecks for online learning algorithms I have observed in my work are the difficulties in (1) learning structured representations of the world that make it more efficient for robots to identify task relevant quantities in the environment and (2) actually executing such algorithms in the real world without continuously reaching states which require human intervention due to slow task progress. The algorithms presented in this dissertation make some steps towards addressing each of these challenges, but there are a number of future directions to explore that I believe would significantly broaden the applicability and utility of online robot learning algorithms.

For the first challenge, I have developed a number of algorithms which leverage offline data to learn data-driven subsets of the environment to restrict exploration for safe and efficient learning (Chapters 5-10). However, moving forward, it would be exciting to explore ways to use such datasets to learn control aware representations in order to facilitate rapid and safe online exploration in environments with high-dimensional observations. There has been a lot of focus in prior work on designing new algorithms for online robot learning, but less focus on studying the impact of the representations on which these algorithms operate. Learning representations that maintain invariance to task-irrelevant distractors and focus on the information needed to plan actions in the environment would be very useful in ensuring that learned robotic control policies are robust and can be learned efficiently. One interesting direction here is to learn representations that are predictive of the types of actions an expert demonstrator might take in the world, but do not necessarily contain the information to reconstruct everything in the environment. Such representations will likely be quite useful for robotic control, as they will contain the information needed to plan actions based on environment observations, but not contain spurious information that is irrelevant for control.

For the second challenge, I have explored algorithms for humans to supervise robots during online exploration so that they can intervene when robots reach states in which they are unlikely to make task progress in Chapters 1-7. I strongly believe that developing more sophisticated algorithms to scalably leverage human resources during robot learning will be of vital importance in industrial applications of robot learning algorithms. When learning new tasks online, learned policies will almost certainly reach states in which they are unable to make further progress or violate safety constraints. This motivates designing algorithms which can allow a large number of robots to query a small number of human supervisors for assistance when safety violations are likely or progress is stalled. This will involve learning metrics which capture task progress as in Chapter 7 and probability of constraint violation as in Chapter 9 and finding intelligent ways to combine these metrics to prioritize different robots for human intervention. This general setting would be particularly interesting in the context of autonomous driving or warehouse automation, in which there are often large numbers of autonomous agents operating in the environment and only a small number of human supervisors available to serve intervention requests, making it critical that human labor is allocated intelligently across the robotic agents. Studying the tradeoffs between different forms of human feedback for robot learning with respect to learning utility and human cost would also be exciting to study moving forward.

Bibliography

[1] Sudeep Dasari et al. "RoboNet: Large-Scale Multi-Robot Learning". In: (2019).

[2] Frederik Ebert et al. "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control". In: *arXiv preprint arXiv:1812.00568* (2018).

[3] Eric Jang et al. "BC-Z: Zero-Shot Task Generalization with Robotic Imitation Learning". In: (2021).

[4] Michael Janner, Qiyang Li, and Sergey Levine. "Offline Reinforcement Learning as One Big Sequence Modeling Problem". In: (2021).

[5] Lili Chen et al. "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: (2021).

[6] Aviral Kumar et al. *Conservative Q-Learning for Offline Reinforcement Learning.* 2020. arXiv: `2006.04779 [cs.LG]`.

[7] Aviral Kumar et al. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction". In: (2019).

[8] 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.

[9] Dean A Pomerleau. "Alvinn: An autonomous land vehicle in a neural network". In: *Conference on Neural Information Processing Systems (NeurIPS).* 1989.

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

[11] 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.

[12] Michael Kelly et al. "HG-DAgger: Interactive Imitation Learning with Human Experts". In: *IEEE International Conference on Robotics and Automation (ICRA).* 2019.

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

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

[15] Takayuki Osa et al. "An algorithmic perspective on imitation learning". In: *arXiv preprint arXiv:1811.06711* (2018).

[16] Jonathan Spencer et al. "Learning from Interventions: Human-robot Interaction as both Explicit and Immplicit Feedback". In: *Robotics: Science and Systems (RSS)*. 2020.

[17] Gokul Swamy et al. "Scaled autonomy: Enabling human operators to control robot fleets". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 5942–5948.

[18] 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.

[19] John Launchbury. "A Natural Semantics for Lazy Evaluation". In: POPL '93. Charleston, South Carolina, USA: Association for Computing Machinery, 1993, pp. 144–154. ISBN: 0897915607. DOI: `10.1145/158511.158618`. URL: `https://doi.org/10.1145/158511.158618`.

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

[21] Auke Jan Ijspeert et al. "Dynamical movement primitives: learning attractor models for motor behaviors". In: *Neural computation* 25.2 (2013).

[22] Alexandros Paraschos et al. "Probabilistic movement primitives". In: *Advances in neural information processing systems*. 2013, pp. 2616–2624.

[23] 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.

[24] 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.

[25] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2008.

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

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

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

[29] Daniel S. Brown et al. “Extrapolating Beyond Suboptimal Demonstrations via Inverse Reinforcement Learning from Observations”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*. 2019.

[30] Daniel S Brown et al. “Safe Imitation Learning via Fast Bayesian Reward Inference from Preferences”. In: *International Conference on Machine Learning*. 2020, pp. 1165–1177.

[31] Akanksha Saran et al. “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.

[32] Hsiao-Yu Tung et al. “Reward learning from narrated demonstrations”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 7004–7013.

[33] Michael Laskey et al. “DART: Noise Injection for Robust Imitation Learning”. In: *Conference on Robot Learning (CoRL)*. Vol. abs/1703.09327. 2017. arXiv: `1703.09327`. URL: `http://arxiv.org/abs/1703.09327`.

[34] Ashwin Balakrishna* et al. “On-Policy Robot Imitation Learning from a Converging Supervisor”. In: *Conference on Robot Learning (CoRL)*. PMLR. 2019.

[35] Michael Laskey et al. “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.

[36] 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.

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

[38] Dorsa Sadigh et al. “Active Preference-Based Learning of Reward Functions”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2017.

[39] Paul F Christiano et al. “Deep reinforcement learning from human preferences”. In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[40] Borja Ibarz et al. “Reward learning from human preferences and demonstrations in Atari”. In: *Advances in Neural Information Processing Systems*. 2018.

[41] Malayandi Palan et al. “Learning Reward Functions by Integrating Human Demonstrations and Preferences”. In: *Proceedings of Robotics: Science and Systems (RSS)*. 2019.

[42] Erdem Bıyık et al. “Asking easy questions: A user-friendly approach to active reward learning”. In: *arXiv preprint arXiv:1910.04365* (2019).

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

[44] Linhai Xie et al. “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.

[45] Andrey Kurenkov et al. “AC-Teach: A Bayesian Actor-Critic Method for Policy Learning with an Ensemble of Suboptimal Teachers”. In: *Conference on Robot Learning (CoRL)*. 2019.

[46] Brijen Thananjeyan* et al. “Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones”. In: *Robotics and Automation Letters (RAL)*. IEEE. 2021.

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

[48] William Saunders et al. “Trial without Error: Towards Safe RL with Human Intervention”. In: *17th International Conference on Autonomous Agents and MultiAgent Systems* (2018).

[49] Fan Wang et al. “Intervention Aided Reinforcement Learning for Safe and Practical Policy Optimization in Navigation”. In: *Conference on Robot Learning (CoRL)*. 2018.

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

[51] Ajay Mandlekar et al. *Human-in-the-Loop Imitation Learning using Remote Teleoperation*. 2020. arXiv: `2012.06733 [cs.RO]`.

[52] Ofra Amir et al. “Interactive Teaching Strategies for Agent Training”. In: *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*. 2016.

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

[54] Jacob W Crandall et al. “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.

[55] 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.

[56] Taylor Kessler Faulkner et al. “Active attention-modified policy shaping: socially interactive agents track”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. 2019.

[57] 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.

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

[59] 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). DOI: `10.1109/41.103436`.

[60] Daniel Seita et al. “Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[61] Emanuel Todorov, Tom Erez, and Yuval Tassa. “MuJoCo: A Physics Engine for Model-Based Control”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2012, pp. 5026–5033. URL: `https://ieeexplore.ieee.org/abstract/document/6386109/`.

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

[63] Ryan Hoque et al. “VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation”. In: *Robotics: Science and Systems (RSS)*. 2020.

[64] Yunpeng Pan et al. “Agile Autonomous Driving using End-to-End Deep Imitation Learning”. In: *Robotics: Science and Systems (RSS)*. 2018.

[65] Felipe Codevilla et al. “End-to-end driving via conditional imitation learning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4693–4700.

[66] Chelsea Finn et al. “One-Shot Visual Imitation Learning via Meta-Learning”. In: *Conference on Robot Learning (CoRL)* (2017).

[67] Bin Fang et al. “Survey of imitation learning for robotic manipulation”. In: *International Journal of Intelligent Robotics and Applications* 3.4 (2019), pp. 362–369.

[68] Aditya Ganapathi et al. “Learning Dense Visual Correspondences in Simulation to Smooth and Fold Real Fabrics”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[69] Priya Sundaresan et al. “Learning Interpretable and Transferable Rope Manipulation Policies Using Depth Sensing and Dense Object Descriptors”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2020.

[70] Oliver Kroemer, Scott Niekum, and George Konidaris. “A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms.” In: *J. Mach. Learn. Res.* 22 (2021), pp. 30–1.

[71] Alex Irpan. *Deep Reinforcement Learning Doesn’t Work Yet.* `https://www.alexirpan.com/2018/02/14/rl-hard.html`. 2018.

[72] Ryan Hoque et al. “LazyDAgger: Reducing context switching in interactive imitation learning”. In: *International Conference on Automation Sciences and Engineering (CASE)*. 2021.

[73] Jessie YC Chen, Michael J Barnes, and Michelle Harper-Sciarini. “Supervisory control of multiple robots: Human-performance issues and user-interface design”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 41.4 (2010), pp. 435–454.

[74] Dmitry Kalashnikov et al. “QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation”. In: *Conference on Robot Learning (CoRL)* (2018).

[75] Felipe Codevilla et al. “Exploring the Limitations of Behavior Cloning for Autonomous Driving”. In: *International Conference on Computer Vision* (2019).

[76] Stephane Ross and J. Andrew Bagnell. *Reinforcement and Imitation Learning via Interactive No-Regret Learning*. 2014. arXiv: `1406.5979 [cs.LG]`.

[77] Yuke Zhu et al. “Robosuite: A Modular Simulation Framework and Benchmark for Robot Learning”. In: *arXiv preprint arXiv:2009.12293*. 2020.

[78] Sandra G Hart. “NASA-task load index (NASA-TLX); 20 years later”. In: *Proceedings of the human factors and ergonomics society annual meeting*. Vol. 50. Sage publications Sage CA: Los Angeles, CA. 2006, pp. 904–908.

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

[80] Yuxuan Liu et al. “Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[81] Yuxuan Liu et al. “Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2018).

[82] Tianhao Zhang et al. “Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2018).

[83] Yixin Gao et al. “JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS) : A Surgical Activity Dataset for Human Motion Modeling”. In: *MICCAI Workshop*. 2014.

[84] Gregory Kahn et al. “PLATO: Policy learning using adaptive trajectory optimization”. In: *IEEE International Conference on Robotics and Automation (ICRA)* (2017), pp. 3342–3349.

[85] Yunpeng Pan et al. “Agile off-road autonomous driving using end-to-end deep imitation learning”. In: *arXiv preprint arXiv:1709.07174* (2017).

[86] Kurtland Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Conference on Neural Information Processing Systems (NeurIPS)* (2018).

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

[88] Brijen Thananjeyan et al. "Extending Deep Model Predictive Control with Safety Augmented Value Estimation from Demonstrations". In: *CoRR* (2019).

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

[90] David Silver et al. "Mastering the game of go without human knowledge". In: *Nature* (2017).

[91] Wen Sun et al. "Dual policy iteration". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2018.

[92] Wen Sun et al. "Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction". In: *International Conference on Machine Learning (ICML)*. 2017.

[93] Jonathan Lee et al. "A Dynamic Regret Analysis and Adaptive Regularization Algorithm for On-Policy Robot Imitation Learning". In: *WAFR* (2019).

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

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

[96] Michael Laskey et al. "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[97] Ching-An Cheng et al. "Online Learning with Continuous Variations: Dynamic Regret and Reductions". In: *CoRR* (2019).

[98] Alexis Jacq et al. "Learning from a Learner". In: *International Conference on Machine Learning (ICML)*. 2019.

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

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

[101] John Schulman et al. "Trust region policy optimization". In: *International Conference on Machine Learning (ICML)*. 2015.

[102] Thanard Kurutach et al. “Model-Ensemble Trust-Region Policy Optimization”. In: *International Conference on Learning Representations (ICLR)*. 2018.

[103] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International Conference on Machine Learning (ICML)* (2018).

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

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

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

[107] Anusha Nagabandi et al. “Deep Dynamics Models for Learning Dexterous Manipulation”. In: *Conference on Robot Learning (CoRL)*. 2019.

[108] Ashwin Balakrishna et al. “On-Policy Robot Imitation Learning from a Converging Supervisor”. In: *Conference on Robot Learning (CoRL)*. 2019.

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

[110] 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.

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

[112] Ugo Rosolia and Francesco Borrelli. “Learning how to autonomously race a car: a predictive control approach”. In: *Proceedings 2017 IFAC World Congress*. IEEE, 2019.

[113] Anil Aswani et al. “Provably Safe and Robust Learning-Based Model Predictive Control”. In: *Automatica* 49 (2011).

[114] Jus Kocijan et al. “Gaussian process model based predictive control”. In: 2004.

[115] Torsten Koller et al. “Learning-based Model Predictive Control for Safe Exploration and Reinforcement Learning”. In: 2018.

[116] Lukas Hewing, Alexander Liniger, and Melanie Zeilinger. “Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars”. In: 2018.

[117] Enrico Terzi et al. "Learning-based predictive control for linear systems: A unitary approach". In: *Automatica* 108 (2019).

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

[119] Juš Kocijan et al. "Gaussian process model based predictive control". In: *Proceedings of the 2004 American Control Conference.*

[120] Marko Bacic et al. "General interpolation in MPC and its advantages". In: *IEEE Transactions on Automatic Control* 48 (2003).

[121] 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.

[122] 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.

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

[124] Kendall Lowrey et al. "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control". In: *International Conference on Machine Learning (ICML).* 2019.

[125] Carlos Florensa et al. "Reverse Curriculum Generation for Reinforcement Learning". In: *Conference on Robot Learning (CoRL).* Vol. abs/1707.05300. 2017. arXiv: `1707.05300`. URL: `http://arxiv.org/abs/1707.05300`.

[126] Cinjon Resnick et al. "Backplay: "Man muss immer umkehren"". In: *CoRR* abs/1807.06919 (2018). arXiv: `1807.06919`. URL: `http://arxiv.org/abs/1807.06919`.

[127] 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.

[128] Boris Ivanovic et al. "BaRC: Backward Reachability Curriculum for Robotic Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA).* 2019.

[129] Ashvin V Nair et al. "Visual Reinforcement Learning with Imagined Goals". In: *Conference on Neural Information Processing Systems (NeurIPS).* 2018.

[130] Tom Schaul et al. "Universal Value Function Approximators". In: *International Conference on Machine Learning (ICML).* 2015.

[131] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems.* 2017, pp. 5048–5058.

[132] 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.

[133] Alex Pui-wai Lee et al. "Gaussian Belief Space Planning for Imprecise Articulated Robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2013.

[134] Hanna Kurniawati et al. "Motion planning under uncertainty for robotic tasks with long time horizons". In: *International Journal of Robotics Research (IJRR)* 30.3 (2011), pp. 308–323.

[135] Zdravko I. Botev et al. *The Cross-Entropy Method for Optimization*. 2013.

[136] Atsushi Sakai et al. "PythonRobotics: a Python code collection of robotics algorithms". In: (2018).

[137] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

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

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

[140] 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)*. 2016.

[141] Dario Amodei et al. "Concrete problems in AI safety". In: *arXiv preprint arXiv:1606.06565* (2016). arXiv: `1606.06565 [cs.AI]`.

[142] D. Seita et al. "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.

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

[144] Kendall Lowrey et al. "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control". In: *International Conference on Machine Learning (ICML)*. 2019.

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

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

[147] Ashvin Nair et al. "Overcoming Exploration in Reinforcement Learning with Demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2018).

[148] 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).

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

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

[151] 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).

[152] Javier García and Fernando Fernández. "A Comprehensive Survey on Safe Reinforcement Learning". In: *Journal of Machine Learning Research (JMLR)* (2015).

[153] 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.

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

[155] Joshua Achiam et al. "Constrained policy optimization". In: *Journal of Machine Learning Research (JMLR)*. 2017.

[156] Felix Berkenkamp et al. "Safe Model-based Reinforcement Learning with Stability Guarantees". In: *NIPS*. 2017.

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

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

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

[160] Matthias Plappert et al. "Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research". In: *CoRR* abs/1802.09464 (2018). arXiv: `1802.09464`. URL: `http://arxiv.org/abs/1802.09464`.

[161] Jur Van Den Berg et al. "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010.

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

[163] Danijar Hafner et al. "Learning Latent Dynamics for Planning from Pixels". In: *International Conference on Machine Learning (ICML)* (2019).

[164] Ryan Hoque et al. "Visuospatial foresight for multi-step, multi-task fabric manipulation". In: *Robotics: Science and Systems (RSS)* (2020).

[165] 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.

[166] 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).

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

[168] Karl Pertsch et al. "Long-horizon visual planning with goal-conditioned hierarchical predictors". In: *Conference on Neural Information Processing Systems (NeurIPS)* (2020).

[169] Stephen Tian et al. "Model-Based Visual Planning with Self-Supervised Functional Distances". In: *International Conference on Learning Representations (ICLR)* (2021).

[170] Tuomas Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In.

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

[172] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *Journal of Machine Learning Research (JMLR)* (2016).

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

[174] Vitchyr H Pong et al. "Skew-fit: State-covering self-supervised reinforcement learning". In: *International Conference on Machine Learning (ICML)* (2020).

[175] 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.

[176] Brijen Thananjeyan* et al. "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.

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

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

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

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

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

[182] Gerrit Schoettler et al. "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).

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

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

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

[186] Aravind Srinivas et al. "Universal Planning Networks". In: *International Conference on Machine Learning (ICML)* (Apr. 2018).

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

[188] Irina Higgins et al. "beta-vae: Learning basic visual concepts with a constrained variational framework". In: *International Conference on Learning Representations (ICLR)* (2017).

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

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

[191] K. Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Conference on Neural Information Processing Systems (NeurIPS)*. 2018.

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

[193] Tuomas Haarnoja et al. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor". In: *International Conference on Machine Learning (ICML)* (2018).

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

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

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

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

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

[199] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: `1707.06347 [cs.LG]`.

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

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

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

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

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

[205] Brijen Thananjeyan et al. *Safety Augmented Value Estimation from Demonstrations (SAVED): Safe Deep Model-Based RL for Sparse Cost Robotic Tasks*. 2020. arXiv: `1905.13402 [cs.LG]`.

[206] Brijen Thananjeyan et al. "Recovery rl: Safe reinforcement learning with learned recovery zones". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 4915–4922.

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

[208] Ashvin Nair et al. "Overcoming exploration in reinforcement learning with demonstrations". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6292–6299.

[209] Todd Hester et al. "Deep q-learning from demonstrations". In: *Thirty-second AAAI conference on artificial intelligence*. 2018.

[210] Yang Gao et al. "Reinforcement learning from imperfect demonstrations". In: *arXiv preprint arXiv:1802.05313* (2018).

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

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

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

[214] Mingxuan Jing et al. "Reinforcement learning from imperfect demonstrations under soft expert guidance". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5109–5116.

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

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

[217] Mel Vecerik et al. *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards*. 2018. arXiv: `1707.08817 [cs.AI]`.

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

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

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

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

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

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

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

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

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

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

[228] Tuomas Haarnoja et al. "Soft Actor-Critic Algorithms and Applications". In: *CoRR* (2018).

[229] Anusha Nagabandi et al. "Deep Dynamics Models for Learning Dexterous Manipulation". In: *Conference on Robot Learning (CoRL)* (2019).

[230] Jaime F. Fisac et al. "A General Safety Framework for Learning-Based Control in Uncertain Robotic Systems". In: *IEEE Transactions on Automatic Control*. 2018.

[231] 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.

[232] 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.

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

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

[235] Yun Shen et al. "Risk-sensitive Reinforcement Learning". In: *Neural Computation*. Vol. 26. 2014.

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

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

[238] Benjamin Eysenbach et al. "Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning". In: *International Conference on Learning Representations* (2018).

[239] Jaime F. Fisac et al. "Bridging Hamilton-Jacobi Safety Analysis and Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

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

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

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

[243] Y. Chow et al. “A Lyapunov-based Approach to Safe Reinforcement Learning”. In: *NeurIPS*. 2018.

[244] Y. Chow et al. “Lyapunov-based Safe Policy Optimization for Continuous Control”. In: *ICML Workshop RL4RealLife*. 2019.

[245] Mohammed Alshiekh et al. “Safe Reinforcement Learning via Shielding”. In: 2018.

[246] 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).

[247] Eitan Altman. *Constrained Markov Decision Processes*. 1999, p. 260. DOI: `10.1016/0167-6377(96)00003-X`.

[248] Minho Hwang et al. “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).

[249] Homanga Bharadhwaj et al. “Conservative Safety Critics for Exploration”. In: *arXiv preprint arXiv:2010.14497* (2020).

[250] Jesse Zhang et al. “Cautious Adaptation for Reinforcement Learning in Safety-Critical Settings”. In: *International Conference on Machine Learning (ICML)* (2020).

[251] Xingyou Song et al. “Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020).

[252] Julian Ibarz et al. “How to Train Your Robot with Deep Reinforcement Learning: Lessons we Have Learned”. In: *International Journal of Robotics Research (IJRR)* (2021).

[253] Eric Mitchell et al. *Offline Meta-Reinforcement Learning with Advantage Weighting*. 2020. arXiv: `2008.06043 [cs.LG]`.

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

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

[256] Yinlam Chow et al. “Risk-Constrained Reinforcement Learning with Percentile Risk Criteria”. In: *Journal of Machine Learning Research* 1 (2015), pp. 1–49.

[257] Homanga Bharadhwaj et al. “Conservative Safety Critics for Exploration”. In: *arXiv preprint arXiv:2010.14497* (2020).

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

[259] Felix Berkenkamp, Angela P. Schoellig, and Andreas Krause. “Safe Controller Optimization for Quadraotors with Gaussian Processes”. In: *Conference on Decision and Control* (2015).

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

[261] 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.

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

[263] Mo Chen et al. “Safe Platooning of Unmanned Aerial Vehicles via Reachability”. In: *Conference on Decision and Control* (2015).

[264] Susmit Jha et al. “Safe Autonomy Under Perception Uncertainty Using Chance-Constrained Temporal Logic”. In: *Transactions on Automated Reasoning* (2017).

[265] 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. URL: `http://www.idsia.ch/~juergen/diploma.html`.

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

[267] 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.

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

[269] 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.

[270] Yan Duan et al. *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning.* 2016. arXiv: `1611.02779 [cs.AI]`.

[271] Jane X Wang et al. *Learning to reinforcement learn.* 2017. arXiv: `1611.05763 [cs.LG]`.

[272] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.* 2017. arXiv: `1703.03400 [cs.LG]`.

[273] Nikhil Mishra et al. *A Simple Neural Attentive Meta-Learner*. 2018. arXiv: `1707.03141 [cs.AI]`.

[274] Rein Houthooft et al. *Evolved Policy Gradients*. 2018. arXiv: `1802.04821 [cs.LG]`.

[275] Kate Rakelly et al. *Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables*. 2019. arXiv: `1903.08254 [cs.LG]`.

[276] Jan Humplik et al. *Meta reinforcement learning as task inference*. 2019. arXiv: `1905.06424 [cs.LG]`.

[277] Rasool Fakoor et al. *Meta-Q-Learning*. 2020. arXiv: `1910.00125 [cs.LG]`.

[278] 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. URL: `https://arxiv.org/abs/1803.07551`.

[279] Anusha Nagabandi et al. *Learning to Adapt in Dynamic, Real-World Environments Through Meta-Reinforcement Learning*. 2019. arXiv: `1803.11347 [cs.LG]`.

[280] Bradly Stadie et al. "The Importance of Sampling inMeta-Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: `https://proceedings.neurips.cc/paper/2018/file/d0f5722f11a0cc839fa2ca6ea49d8585-Paper.pdf`.

[281] Jin Zhang et al. *Learn to Effectively Explore in Context-Based Meta-RL*. June 2020.

[282] Evan Zheran Liu et al. *Decoupling Exploration and Exploitation for Meta-Reinforcement Learning without Sacrifices*. 2021. arXiv: `2008.02790 [cs.LG]`.

[283] 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. URL: `https://openreview.net/forum?id=8cpHIfgY4Dj`.

[284] Karol Arndt et al. "Meta Reinforcement Learning for Sim-to-real Domain Adaptation". In: *IEEE International Conference on Robotics and Automation (ICRA)* (2020).

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

[286] Elon Rimon and Joel Burdick. *The Mechanics of Robot Grasping*. Cambridge University Press, 2019.

[287] Antonio Bicchi and Vijay Kumar. "Robotic grasping and contact: A review". In: *IEEE International Conference on Robotics and Automation (ICRA)*. Vol. 1. IEEE. 2000, pp. 348–353.

[288] Domenico Prattichizzo and Jeffrey C Trinkle. "Grasping". In: *Springer handbook of robotics*. Springer, 2016, pp. 955–988.

[289] Richard M Murray. *A mathematical introduction to robotic manipulation.* CRC press, 2017.

[290] David Wang et al. "Adversarial Grasp Objects". In: *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE).* IEEE. 2019, pp. 241–248.

[291] Ian Lenz, Honglak Lee, and Ashutosh Saxena. "Deep learning for detecting robotic grasps". In: *International Journal of Robotics Research (IJRR)* 34.4-5 (2015), pp. 705–724.

[292] Daniel Kappler, Jeannette Bohg, and Stefan Schaal. "Leveraging big data for grasp planning". In: *IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2015, pp. 4304–4311.

[293] Sergey Levine et al. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *International Journal of Robotics Research (IJRR)* 37.4-5 (2018), pp. 421–436.

[294] Jeffrey Mahler et al. "Learning ambidextrous robot grasping policies". In: *Science Robotics* 4.26 (2019), eaau4984.

[295] Stephen James et al. "Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* 2019, pp. 12627–12637.

[296] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach". In: *Proc. of Robotics: Science and Systems (RSS).* 2018.

[297] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2016, pp. 3406–3413.

[298] Michael Laskey et al. "Multi-armed bandit models for 2d grasp planning with uncertainty". In: *2015 IEEE International Conference on Automation Science and Engineering (CASE).* IEEE. 2015, pp. 572–579.

[299] Jeffrey Mahler et al. "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards". In: *2016 IEEE International Conference on Robotics and Automation (ICRA).* IEEE. 2016, pp. 1957–1964.

[300] Jeffrey Mahler et al. "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *Robotics: Science and Systems (RSS).* 2018.

[301] Min Liu et al. "Deep Differentiable Grasp Planner for High-DOF Grippers". In: *ArXiv* abs/2002.01530 (2020).

[302] Jeannette Bohg et al. "Data-driven grasp synthesis—a survey". In: *IEEE Transactions on Robotics* 30.2 (2013), pp. 289–309.

[303] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. "Robotic grasping of novel objects using vision". In: *International Journal of Robotics Research (IJRR)* 27.2 (2008), pp. 157–173.

[304] Cristian Bodnar et al. "Quantile QT-Opt for Risk-Aware Vision-Based Robotic Grasping". In: *ArXiv* abs/1910.02787 (2019).

[305] Edward Johns, Stefan Leutenegger, and Andrew J Davison. "Deep learning a grasp function for grasping under gripper pose uncertainty". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 4461–4468.

[306] Ulrich Viereck et al. "Learning a visuomotor controller for real world robotic grasping using simulated depth images". In: *arXiv preprint arXiv:1706.04652* (2017).

[307] Konstantinos Bousmalis et al. "Using simulation and domain adaptation to improve efficiency of deep robotic grasping". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[308] Jeffrey Mahler et al. "Dex-Net 3.0: Computing Robust Robot Vacuum Suction Grasp Targets in Point Clouds using a New Analytic Model and Deep Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[309] Ole-Magnus Pedersen, Ekrem Misimi, and François Chaumette. "Grasping Unknown Objects by Coupling Deep Reinforcement Learning, Generative Adversarial Networks, and Visual Servoing". In: *ICRA 2020 - IEEE International Conference on Robotics and Automation*. Paris, France: IEEE, May 2020, pp. 1–8. URL: `https://hal.inria.fr/hal-02495837`.

[310] Daniel J. Russo et al. "now publishers - A Tutorial on Thompson Sampling". In: 11.1 (2018), pp. 1–96.

[311] Olivier Chapelle and Lihong Li. "An Empirical Evaluation of Thompson Sampling". In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor et al. Curran Associates, Inc., 2011, pp. 2249–2257. URL: `http://papers.nips.cc/paper/4321-an-empirical-evaluation-of-thompson-sampling.pdf`.

[312] Maurice G Kendall. "A new measure of rank correlation". In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[313] Maurice G Kendall. "The treatment of ties in ranking problems". In: *Biometrika* 33.3 (1945), pp. 239–251.

[314] Nicolas Mellado, Dror Aiger, and Niloy J. Mitra. *Super 4PCS Library*. https://github.com/nmellado/ 2017.

[315] Douglas Morrison, Peter Corke, and Jürgen Leitner. "Learning robust, real-time, reactive robotic grasping". In: *International Journal of Robotics Research (IJRR)* 39.2-3 (2020), pp. 183–201.

[316] Junggon Kim et al. “Physically based grasp quality evaluation under pose uncertainty”. In: *IEEE Transactions on Robotics* 29.6 (2013), pp. 1424–1439.

[317] Changhyun Choi et al. “Learning object grasping for soft robot hands”. In: *IEEE Robotics and Automation Letters (RA-L)* 3.3 (2018), pp. 2370–2377.

[318] Michel Breyer et al. “Comparing task simplifications to learn closed-loop object picking using deep reinforcement learning”. In: *IEEE Robotics and Automation Letters (RA-L)* 4.2 (2019), pp. 1549–1556.

[319] Oliver Kroemer, Scott Niekum, and George Konidaris. “A review of robot learning for manipulation: Challenges, representations, and algorithms”. In: *arXiv preprint arXiv:1907.03146* (2019).

[320] Kate Sanders et al. “Non-Markov Policies to Reduce Sequential Failures in Robot Bin Picking”. In: *IEEE Conference on Automation Science and Engineering (CASE)*. 2020.

[321] Michael Laskey et al. “Budgeted multi-armed bandit models for sample-based grasp planning in the presence of uncertainty”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[322] John Oberlin and Stefanie Tellex. “Autonomously acquiring instance-based object models from experience”. In: *International Symposium on Robotics Research (ISRR)*. Springer, 2015, pp. 73–90.

[323] Katherine Li et al. “Accelerating Grasp Exploration by Leveraging Learned Priors”. In: *IEEE Conference on Automation Science and Engineering (CASE)*. 2020.

[324] Oliver B Kroemer et al. “Combining active learning and reactive control for robot grasping”. In: *Robotics and Autonomous systems* 58.9 (2010), pp. 1105–1116.

[325] Clemens Eppner and Oliver Brock. “Visual detection of opportunities to exploit contact in grasping using contextual multi-armed bandits”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[326] Qingkai Lu, Mark Van der Merwe, and Tucker Hermans. “Multi-Fingered Active Grasp Learning”. In: *arXiv preprint arXiv:2006.05264* (2020).

[327] Ken Goldberg et al. “Part pose statistics: Estimators and experiments”. In: *IEEE Transactions on Robotics and Automation* 15.5 (1999), pp. 849–857.

[328] Ken Goldberg. “Orienting Polygonal Parts without Sensors”. In: *Algorithmica* (1993).

[329] Christopher Correa et al. “Robust Toppling for Vacuum Suction Grasping”. In: (2019).

[330] Kenneth Y Goldberg and Matthew T Mason. “Bayesian grasping”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 1990, pp. 1264–1269.

[331] Mark Moll and Michael A Erdmann. “Manipulation of pose distributions”. In: *International Journal of Robotics Research (IJRR)* 21.3 (2002), pp. 277–292.

[332] A. S. Rao and K. Y. Goldberg. "Manipulating algebraic parts in the plane". In: *IEEE Transactions on Robotics and Automation* 11.4 (1995), pp. 598–602.

[333] Thomas Jaksh, Ronald Ortner, and Peter Auer. "Near-optimal Regret Bounds for Reinforcement Learning". In: *Journal of Machine Learning Research (JMLR)*. 2010.

[334] Aristide Tossou, Debabrota Basu, and Christos Dimitrakakis. "Near-optimal Optimistic Reinforcement Learning using empirical Bernstein Inequalities". In: 2019.

[335] Sarah Filippi, Olivier Cappé, and Aurélien Garivier. "Optimism in Reinforcement Learning and Kullback-Leibler Divergence". In: *Annual Allerton Conference on Communication, Control, and Computing*. 2010.

[336] Ian Osband and Benjamin Van Roy. "Near-optimal Reinforcement Learning in Factored MDPs". In: *Neural Information Processing Systems (NeurIPS)*. 2014.

[337] Haiyan Chen and Fuji Zhang. "The expected hitting times for finite Markov chains". In: *Linear Algebra and its Applications* 428.11-12 (2008), pp. 2730–2749.

[338] Ziping Xu and Ambuj Tewari. "Near-optimal Reinforcement Learning in Factored MDPs: Oracle-Efficient Algorithms for the Non-episodic Setting". In: *International Conference on Machine Learning (ICML)*. 2018.

[339] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem". In: *Machine learning* 47.2-3 (2002), pp. 235–256.

[340] Shipra Agrawal and Navin Goyal. "Further Optimal Regret Bounds For Thompson Sampling". In: *Sixteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2013.

[341] Douglas Morrison, Peter Corke, and Jurgen Leitner. "EGAD! an Evolved Grasping Analysis Dataset for diversity and reproducibility in robotic manipulation". In: *IEEE Robotics and Automation Letters (RA-L)* (2020).

[342] Ronan Fruit. *Exploration-Exploitation in Reinforcement Learning*. `https://github.com/RonanFR/UCRL`. 2018.

[343] Moloud Abdar et al. "A review of uncertainty quantification in deep learning: Techniques, applications, and challenges". In: *arXiv preprint arXiv:2011.06225* (2021).

[344] Patrik Puchert et al. "Data-driven deep density estimation". In: *arXiv preprint arXiv:2107.11085* (2021).

[345] Timothy Hospedales et al. "Meta-Learning in Neural Networks: A Survey". In: *arXiv preprint arXiv:2004.05439* (2020).

[346] 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.

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

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

[349] 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.

[350] Pranjal Tandon. *pytorch-soft-actor-critic*. `https://github.com/pranz24/pytorch-soft-actor-critic`. 2018.

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

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

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

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

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

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

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

Appendix A

LazyDAgger: Reducing Context Switching in Interactive Imitation Learning

Here we provide further details on our MuJoCo experiments, hyperparameter sensitivity, simulated fabric experiments, and physical fabric experiments.

A.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 2.3. We provide the state space dimensionality $|S|$, action space dimensionality $|A|$, and LazyDAgger hyperparameters (see Algorithm 1) for each environment in Table A.1. The β_H value in the table is multiplied with the maximum possible action discrepancy $||a_{high} - a_{low}||_2^2$ to become the threshold for training $f(\cdot)$. In MuJoCo environments, $a_{high} = \vec{1}$ and $a_{low} = -\vec{1}$. The β_H value used for SafeDAgger in all experiments is chosen by the method provided in the chapter introducing SafeDAgger [11]: 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 A.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.

A.2 LazyDAgger Switching Thresholds

As described in Section 2.4.1, 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 and Cho [11]; 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 A.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.

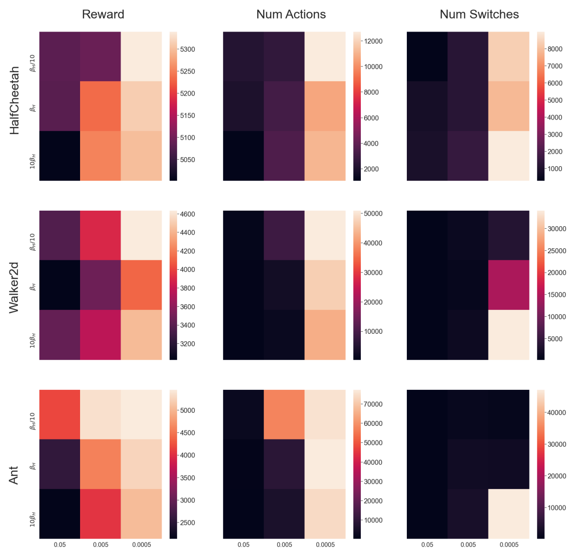


Figure A.1: LazyDAgger β_R and β_H sensitivity heatmaps across the 3 MuJoCo environments. The x-axis denotes β_H and the y-axis denotes β_R. Note that β_R is a function of β_H. Each of the 3 environments was run 9 times with the different settings of β_R and β_H. As in Figure 2.3 we plot test reward, number of supervisor actions, and number of context switches.

A.3 Fabric Smoothing in Simulation

Fabric Simulator

More information about the fabric simulator can be found in Seita et al. [60], 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 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 2.4 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 [60] 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 [60], 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 [60] 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 2.5.2.

Actor Policy and Discrepancy Classifier

The actor policy is a convolutional neural network with the same architecture as [60], 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 [60]. To further accelerate training, we parallelize environment interaction across 20 CPUs, and before

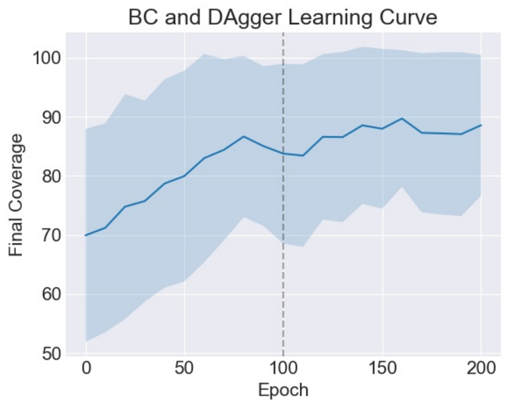


Figure A.2: Behavior Cloning and DAgger performance across 10 test episodes evaluated every 10 epochs. Shading indicates 1 standard deviation. The first 100 epochs (left half) are Behavior Cloning epochs and the second 100 (right half) are DAgger epochs.

DAgger iterations we pretrain with 100 epochs of Behavior Cloning on the dataset of 20,232 state-action pairs available at [60]'s project website. Additional training hyperparameters are given in Table A.2 and the learning curve is given in Figure A.2.

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 A.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.

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 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 A.3). Hyperparameters are given in Table A.3.

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 A.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 A.4). Note that the offline data is split 70%/30% as in Section A.1.

A.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 2.1. The gripper is equipped with reverse tweezers for more precise picking of deformable materials. Neural network architecture is consistent with Section A.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 A.4.

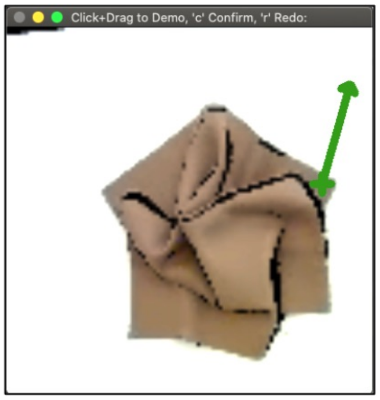


Figure A.3: The user interface for human interventions. The current observation of the fabric state from the robot's perspective is displayed, with an overlaid green arrow indicating the action the human has just specified.

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 2.4). 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 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 A.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
$\|\mathcal{D}\|$	875
$\|\mathcal{D}_S\|$	375
Batch Size	50
Gradient Steps per Epoch	125
π Learning Rate	1e-4
f Learning Rate	1e-3
L2 Regularization	1e-5

Table A.4: Hyperparameters for physical fabric experiments provided in the same format as Table A.3. Since this is at execution time, N, T and σ^2 hyperparameters do not apply.

Appendix B

ThriftyDAgger: Budget-Aware Novelty and Risk Gating for Interactive Imitation Learning

In Appendix B.1, we discuss algorithmic details for ThriftyDAgger and all comparisons. Then, Appendix B.2 discusses implementation and hyperparameter details for all algorithms. In Appendix B.3, we provide additional details about the simulation and physical experiment domains, and in Appendix B.4, we describe the protocol and detailed results from the conducted user study.

B.1 Algorithm Details

Here we provide a detailed algorithmic description of ThriftyDAgger and all comparisons.

B.1.1 ThriftyDAgger

The full pseudocode for ThriftyDAgger is provided in Algorithm 7. ThriftyDAgger first initializes π_r via Behavior Cloning on offline transitions ($\mathcal{D}_h$ from the human supervisor, π_h) (line 1-2). Then, π_r collects an initial offline dataset $\mathcal{D}_r$ from the resulting π_r, initializes $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ by optimizing Equation (3.5) on $\mathcal{D}_r \cup \mathcal{D}_h$, and initializes parameters $\beta_H, \beta_R, \delta_H$, and δ_R as in Section 3.3.4 (lines 3-5). We then collect data for N episodes, each with up to T timesteps. In each timestep of each episode, we determine whether robot policy π_r or human supervisor π_h should be in control using the procedure in Section 3.3.3 (lines 10-20). Transitions in autonomous mode are aggregated into $\mathcal{D}_r$ while transitions in supervisor mode are aggregated into $\mathcal{D}_h$. Episodes are terminated either when the robot reaches a valid goal state or has exhausted the time horizon T. At this point, we re-initialize the policy to autonomous mode and update parameters $\beta_H, \beta_R, \delta_H$, and δ_R as in Section 3.3.4 (lines 21-

23). After each episode, π_r is updated via supervised learning on $\mathcal{D}_h$, while $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ is updated on $\mathcal{D}_r \cup \mathcal{D}_h$ to reflect the task success probability of the resulting π_r (lines 24-26).

B.1.2 Behavior Cloning

We train policy π_R via direct supervised learning with a mean-squared loss to predict reference control actions given a dataset of (state, action) tuples. Behavior Cloning is trained only on full expert demonstrations collected offline from π_H and is not allowed access to online interventions. Thus, Behavior Cloning is trained only on dataset $\mathcal{D}_h$ (line 1, Algorithm 7) and the policy is frozen thereafter. In our simulation experiments, Behavior Cloning is given 50% more offline data than the other algorithms for a more fair comparison, such that the amount of additional offline data is approximately equal to the average amount of online data provided to the other algorithms.

B.1.3 SafeDAgger

SafeDAgger [11] is an interactive imitation learning algorithm which selects between autonomous and supervisor mode using a classifier f that discriminates between "safe" states, for which π_R's proposed action is within some threshold β_H of that proposed by supervisor policy π_H, and "unsafe" states, for which this action discrepancy exceeds β_H. SafeDAgger learns this classifier using dataset $\mathcal{D}_h$ from Algorithm 7, and updates f online as $\mathcal{D}_h$ is expanded through human interventions. During policy rollouts, if f marks a state as safe, the robot policy is executed (autonomous mode), while if f marks a state as unsafe, the supervisor is queried for an action. While this approach can be effective in some domains [11], prior work [72] suggests that this intervention criterion can lead to excessive context switches between the robot and supervisor, and thus impose significant burden on a human supervisor. As in ThriftyDAgger and other DAgger [8] variants, SafeDAgger updates π_R on an aggregated dataset of all transitions collected by the supervisor (analogous to $\mathcal{D}_h$ in Algorithm 7).

B.1.4 LazyDAgger

LazyDAgger [72] builds on SafeDAgger [11] and trains the same action discrepancy classifier f to determine whether the robot and supervisor policies will significantly diverge at a given state. However, LazyDAgger introduces a few modifications to SafeDAgger which lead to lengthier and more informative interventions in practice. First, LazyDAgger observes that when the supervisor has control of the system (supervisor mode), querying f for estimated action discrepancy is no longer necessary since we can simply query the robot policy at any state during supervisor mode to obtain a true measure of the action discrepancy between the robot and supervisor policies. This prevents exploiting approximation errors in f when the supervisor is in control. Second, LazyDAgger introduces an asymmetric switching condition between autonomous and supervisor control, where switches are executed from

Algorithm 7 ThriftyDAgger

```
Require: Number of episodes $N$, time horizon $T$, supervisor policy $\pi_H$, desired context switching rate $\alpha_H$
 1: Collect offline dataset $\mathcal{D}_h$ of $(s, a^H)$ tuples with $\pi_H$
 2: Initialize $\pi_R$ via Behavior Cloning on $\mathcal{D}_h$
 3: Collect offline dataset $\mathcal{D}_r$ of $(s, a^R)$ tuples with $\pi_R$
 4: Initialize $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ by optimizing Equation (3.4) on $\mathcal{D}_r \cup \mathcal{D}_h$
 5: Optimize $\beta_H, \beta_R, \delta_H, \delta_R$ on $\mathcal{D}_h$ {Online tuning based on $\alpha_H$ (Section 3.3.4)}
 6: for $i \in \{1, \dots N\}$ do
 7:     Initialize $s_0$, Mode ← Autonomous
 8:     for $t \in \{1, \dots T\}$ do
 9:         $a^R_t = \pi_R(s_t)$ {Determine control mode (Section 3.3.3)}
10:         if Mode = Supervisor or Intervene($s_t, \delta_H, \beta_H$) then
11:             $a^H_t = \pi_H(s_t)$
12:             $\mathcal{D}_h \leftarrow \mathcal{D}_h \cup \{(s_t, a^H_t)\}$
13:             Execute $a^H_t$ {Default control mode for next timestep (Section 4.3)}
14:             if Cede($s_t, \delta_R, \beta_R$) then
15:                 Mode ← Autonomous
16:             else
17:                 Mode ← Supervisor
18:             end if
19:         else
20:             Execute $a^R_t$
21:             $\mathcal{D}_r \leftarrow \mathcal{D}_r \cup \{(s_t, a_t)\}$
22:         end if
23:         if Terminal state reached then
24:             Exit Loop, Mode ← Autonomous
25:             Recompute $\beta_H, \beta_R, \delta_H$ {Online tuning based on $\alpha_H$ (Section 3.3.4)}
26:         end if
27:     end for
28:     $\pi_R \leftarrow \arg\min_{\pi_R} \mathbb{E}_{(s_t, a^H_t) \sim \mathcal{D}_h} [\mathcal{L}(\pi_R(s_t), \pi_H(s_t))]$
29:     Collect $\mathcal{D}_r$ offline with robot policy $\pi_R$
30:     Update $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ on $\mathcal{D}_r \cup \mathcal{D}_h$ {Update Q-function via Equation (3.6)}
31: end for
```

autonomous to supervisor mode if f indicates that the predicted action discrepancy is above β_H, but switches are only executed from supervisor mode back to autonomous mode if the true action discrepancy is below some value $\beta_R < \beta_H$. This encourages lengthier interventions, leading to fewer context switches between autonomous and supervisor modes. Finally, LazyDAgger injects noise into supervisor actions in order to spread the distribution of states in which reference controls from the supervisor are available. ThriftyDAgger builds on the asymmetric switching criterion introduced by LazyDAgger, but introduces a new switching criterion based on the estimated task success probability, which we found significantly improved performance in practice.

B.1.5 HG-DAgger

Unlike SafeDAgger, LazyDAgger, and ThriftyDAgger, which are robot-gated and autonomously determine when to solicit intervention requests, HG-DAgger is human-gated, and thus requires that the supervisor determine the timing and length of interventions. As in ThriftyDAgger, HG-DAgger updates π_R on an aggregated dataset of all transitions collected by the supervisor (analogous to $\mathcal{D}_h$ in Algorithm 7).

B.2 Hyperparameter and Implementation Details

Here we provide a detailed overview of all hyperparameter and implementation details for ThriftyDAgger and all comparisons to facilitate reproduction of all experiments. We also include code in the supplement, and will release a full open-source codebase after anonymous review.

B.2.1 ThriftyDAgger

Peg Insertion (Simulation): We initially populate $\mathcal{D}_h$ with 2,687 offline transitions, which correspond to 30 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_R. We represent π_R with an ensemble of 5 neural networks, trained on bootstrapped samples of data from $\mathcal{D}_h$ in order to quantify uncertainty for novelty estimation. Each neural network is trained using the Adam Optimizer (learning rate 1e$-$3) with 5 training epochs, 500 gradient steps in each training epoch, and a batch size of 100. All networks consist of 2 hidden layers, each with 256 hidden units with ReLU activations, and a Tanh output activation.

The Q-function used for risk-estimation, $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$, is trained with a batch size of 50, and batches are balanced such that 10% of all sampled transitions contain a state in the goal set. We train $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ with the Adam Optimizer, with a learning rate of 1e$-$3 and discount factor $\gamma = 0.9999$. In order to train $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$, we collect 10 test episodes from π_R every 2,000 environment steps. We represent $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ with a 2 hidden layer neural net in which each hidden layer has 256 hidden units with ReLU activations and with a sigmoid output activation. The state and action are concatenated before they are fed into $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$.

Block Stacking (Simulation): This is an additional simulation environment not included in the main text. Results and a description of the task are in Section B.3.2. We populate $\mathcal{D}_h$ with 1,677 offline transitions, corresponding to 30 task demonstrations, to initialize π_R. All other parameters and implementation details are identical to the peg insertion environment.

Cable Routing (Physical): We initially populate $\mathcal{D}_h$ with 1,381 offline transitions, corresponding to 25 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_R. We again represent π_R with an ensemble of 5 neural networks, trained on bootstrapped samples of data from $\mathcal{D}_h$ in order to quantify uncertainty for novelty estimation. Each neural network is trained using the Adam Optimizer (learning rate 2.5e−4) with 5 training epochs, 300 gradient steps per training epoch, and a batch size of 64. All networks consist of 5 convolutional layers (format: (in_channels, out_channels, kernel_size, stride)): $[(3, 24, 5, 2), (24, 36, 5, 2), ($ followed by 4 fully connected layers (format: (in_units, out_units)): $[(64, 100), (100, 50), (50, 10), (10, 2)]$. Here we utilize ELU (exponential linear unit) activations with a Tanh output activation.

The Q-function used for risk-estimation, $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$, is trained with a batch size of 64 as well, and batches are balanced such that 10% of all sampled transitions contain a state in the goal set. We train $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ with the Adam Optimizer with a learning rate of 2.5e−4 and discount factor $\gamma = 0.9999$. In order to train $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$, we collect 5 test episodes from π_R every 500 environment steps. We represent $\hat{Q}^{\pi_R}_{\phi,\mathcal{G}}$ with a neural network with the same 5 convolutional layers as the policy networks above, but with the fully connected layers as follows (format: (in_units, out_units)): $[(64 + 2, 100), (100, 50), (50, 10), (10, 1)]$. We concatenate the action with the state embedding resulting from the 5 convolutional layers (hence the 64 + 2) and feed the resulting concatenated embedding into the 4 fully connected layers above. We utilize ELU (exponential linear unit) activations with a sigmoid output activation.

B.2.2 Behavior Cloning

Peg Insertion (Simulation): For Behavior Cloning, we initially populate $\mathcal{D}_h$ with 4,004 offline transitions, corresponding to 45 task demonstrations collected by an expert human supervisor, to initialize the robot policy π_R (note that this is more transitions than are provided to ThriftyDAgger). All other details are the same as ThriftyDAgger for training π_R.

Block Stacking (Simulation): We initially populate $\mathcal{D}_h$ with 3,532 offline transitions, corresponding to 60 task demonstrations, to initialize π_R. Note that Behavior Cloning has access to twice as many offline demonstrations as the other algorithms.

Cable Routing (Physical): We train π_R with the same architecture and procedure as for ThriftyDAgger, but only on the initial offline data.

B.2.3 SafeDAgger

We use the same hyperparameters and architecture for training π_R as for ThriftyDAgger. Unlike ThriftyDAgger, SafeDAgger does not have a mechanism to automatically set intervention thresholds when provided an intervention budget α_H. Thus, we must specify a value for the switching threshold β_H. We use $\beta_H = 0.008$, since this is recommended in [11] as the value which was found to work well in experiments (in practice, this value marks about 20% of states as "unsafe").

B.2.4 LazyDAgger

We use the same hyperparameters and architecture for training π_R as for ThriftyDAgger. Unlike ThriftyDAgger, LazyDAgger does not have a mechanism to automatically set intervention thresholds when provided an intervention budget α_H. Thus, we must specify a value for both switching thresholds β_H and β_R. We use $\beta_H = 0.015$, $\beta_R = 0.25\beta_H$ and use a noise covariance matrix of $0.02\mathcal{N}(0, I)$ when injecting noise into the supervisor actions. These values were tuned to strike a balance between supervisor burden and policy performance.

B.2.5 HG-DAgger

All hyperparameters and architectures are identical to those used for Behavior Cloning, without the extra offline demonstrations. Note that for HG-DAgger, the supervisor determines the timing and length of interventions.

B.3 Environment Details and Additional Metrics

B.3.1 Peg Insertion in Simulation

We evaluate our algorithm and baselines in the Robosuite environment (`https://robosuite.ai`) [77], which builds on MuJoCo [61] to provide a standardized suite of benchmark tasks for robot learning. Specifically, we consider the "Nut Assembly" task, in which a robot must grab a ring from a random initial pose and place it over a cylinder at a fixed location. We consider a variant of the task that considers only 1 ring and 1 target, though the simulator allows 2 rings and 2 targets. The states are $s \in \mathbb{R}^{51}$ and actions $a \in \mathbb{R}^5$ (translation in the XY-plane, translation in the Z-axis, rotation around the Z-axis, and opening or closing the gripper). The simulated robot arm is a UR5e, and the controller reaches a commanded pose via operational space control with fixed impedance. To avoid bias due to variable teleoperation speeds and ensure that the Markov property applies, we abstract 10 timesteps in the simulator into 1 environment step, and in supervisor mode we pause the simulation until keyboard input is received. This prevents accidentally collecting "no-op" expert labels and allows the end effector to "settle" instead of letting its momentum carry on to the next state. In practice it does not make the task more difficult, as control is still fine-grained enough for precise manipulation. Each episode is terminated upon successful task completion or when 175 actions are executed. Interventions are collected through a keyboard interface. In Table B.1, we report additional metrics for the peg insertion simulation experiment and find that ThriftyDAgger solicits fewer interventions than prior algorithms at training time while achieving a higher success rate during training than all algorithms other than HG-DAgger, though it does request more human actions. The train success rate column also indicates that ThriftyDAgger achieves *throughput* comparable to HG-DAgger and higher than other baselines, as ThriftyDAgger has more task successes in the same amount of time (10,000 timesteps for all algorithms). At execution time, ThriftyDAgger collects lengthier interven-

Table B.1: **Peg Insertion in Simulation Additional Metrics:** We report additional statistics for the peg insertion task: total number of interventions (T Ints), total number of offline and online human actions (T Acts (H)), and total number of robot actions (T Acts (R))) at training time across all trajectories (successful and unsuccessful). We report these same metrics at execution time, but T Acts (H) does not include offline human actions, as at execution time it does not refer to the number of training samples for the robot policy. We also report the success rate of the mixed control policy at *training* time (Train Succ.). Results suggest that ThriftyDAgger solicits fewer interventions than prior algorithms at training time while achieving a comparable success rate and throughput to HG-DAgger. At execution time, ThriftyDAgger collects lengthier interventions than prior algorithms but succeeds more often at the task (Table 3.1).

Algorithm	Training Interventions			Train Succ.	Execution Interventions		
	T Ints	T Acts (H)	T Acts (R)		T Ints	T Acts (H)	T Acts (R)
Behavior Cloning	N/A	4004	N/A	N/A	N/A	N/A	N/A
SafeDAgger	334	4227	8460	48/73	81	396	1781
LazyDAgger	82	3683	9004	37/67	30	290	2422
HG-DAgger	124	4392	8295	83/83	23	342	2071
Ours (-Novelty)	60	5242	7445	62/80	12	157	2649
Ours (-Risk)	87	3623	9064	72/81	30	237	2255
Ours: ThriftyDAgger	84	6840	5847	76/86	27	426	1696

tions than prior algorithms, but as a result is able to succeed more often at execution time as discussed in the main text.

B.3.2 Block Stacking in Simulation

To further evaluate the algorithm and baselines in simulation, we also consider the block stacking task from Robosuite (see previous section). Here the robot must grasp a cube in a randomized initial pose and place it on top of a second cube in another randomized pose. See Table B.2 for training results and Figure B.1 for an illustration of the experimental setup. Due to the randomized place position, small placement region, and geometric symmetries, the task is more difficult than peg insertion, as evidenced by the lower autonomous success rate for all algorithms. However, we still see that ThriftyDAgger achieves comparable performance to HG-DAgger in terms of autonomous success rate, success rate during training, and throughput, while outperforming the other baselines and ablations. ThriftyDAgger also solicits fewer interventions than prior algorithms, but generally requires more human actions as these interventions tend to be lengthier. This makes ThriftyDAgger well-suited to situations in which the cost of context switches (latency) may be high.

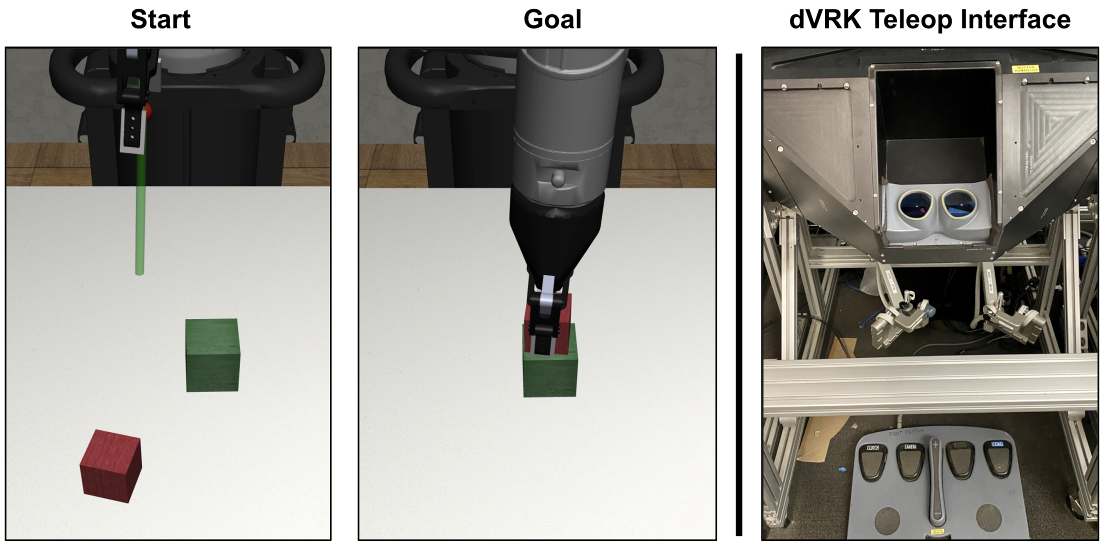


Figure B.1: **Left:** An example start and goal state for the block stacking environment in Robosuite. The goal is to place the red block on top of the green one. Initial poses of both blocks are randomized. **Right:** The da Vinci Research Kit Master Tool Manipulator (MTM) 7DOF interface used to provide human interventions in the physical experiments. The human expert views the workspace through the viewer (top) and teleoperates the robot by moving the right joystick (middle) in free space while pressing the rightmost pedal (bottom).

B.3.3 Physical Cable Routing

Finally, we evaluate our algorithm on a visuomotor cable routing task with a da Vinci Research Kit surgical robot. We take RGB images of the scene with a Zivid One Plus camera inclined at about 45 degrees to the vertical. These images are cropped into a square and downsampled to 64×64 before they are passed to the neural network policy. The cable state is initialized to approximately the same shape (see Figure 3.2) with the cable initialized in the robot's gripper. The workspace is approximately 10 cm $\times$ 10 cm, and each component of the robot action $(\Delta x, \Delta y)$ is at most 1 cm in magnitude. To avoid collision with the 4 obstacles, we implement a "logical obstacle" as 1-cm radius balls around the center of each obstacle. Actions that enter one of these regions are projected to the boundary of the circle. Each episode is terminated upon successful task completion or 100 actions executed. Interventions are collected through a 7DOF teleoperation interface (Figure B.1) that matches the pose of the robot arm, with rotation of the end effector disabled. Teleoperated actions are mapped to the robot's action space by projecting pose deltas to the 2D plane at 1 second intervals. The human teleoperates the robot at a frequency that roughly corresponds to taking actions with the maximum magnitude (1 cm / sec). In Table B.3, we report additional metrics for the physical cable routing experiment and find that ThriftyDAgger solicits a number of interventions similar to HG-DAgger while achieving a similar success rate during training.

Table B.2: **Block Stacking in Simulation Results:** We report the number of interventions (Ints), number of human actions (Acts (H)), and number of robot actions (Acts (R)) during training (over successful trajectories as in Table 3.1) and report the success rate of the robot policy after training when no interventions are allowed (Auto Succ.). We also report the total number of interventions (T Int), total number of actions from the human (offline and online, in T Acts (H)), total number of actions executed by the robot (T Acts (R)), and the success rate of the mixed control policy during training (Train Succ.). Results suggest that ThriftyDAgger achieves comparable performance to HG-DAgger in terms of both autonomous and training success rates while outperforming the other baselines and ablations. ThriftyDAgger also solicits fewer interventions than prior algorithms, but generally requires more human actions.

Algorithm	Ints	Acts (H)	Acts (R)	Auto Succ.	T Ints	T Acts (H)	T Acts (R)	Train Succ.
Behavior Cloning	N/A	N/A	68.0 ± 11.4	5/100	N/A	3532	N/A	N/A
SafeDAgger	5.00 ± 3.41	40.5 ± 14.1	44.3 ± 25.6	3/100	574	4387	7290	27/68
LazyDAgger	1.81 ± 1.02	25.8 ± 17.8	56.6 ± 28.3	40/100	85	2940	8737	36/75
HG-DAgger	1.62 ± 0.91	22.5 ± 16.5	54.6 ± 14.2	**56/100**	201	4535	7142	124/125
Ours (-Novelty)	0.65 ± 0.70	43.7 ± 13.3	28.6 ± 28.5	8/100	37	3599	8078	23/69
Ours (-Risk)	1.89 ± 0.72	12.9 ± 7.7	72.4 ± 25.5	31/100	109	2518	9159	47/79
Ours: ThriftyDAgger	1.33 ± 0.76	35.4 ± 15.8	37.2 ± 27.5	**52/100**	153	5873	5804	111/120

Table B.3: **Physical Cable Routing Additional Metrics:** We report additional statistics for the peg insertion task: total number of interventions (T Ints), total number of offline and online human actions (T Acts (H)), and total number of robot actions (T Acts (R))) at training time across all trajectories. We report these same metrics at execution time, but T Acts (H) does not include offline human actions, as at execution time it does not refer to the number of training samples for the robot policy. We also report the success rate of the mixed control policy at *training* time (Train Succ.). Results suggest that ThriftyDAgger needs fewer interventions than HG-DAgger while achieving a similar training success rate. At execution time, we find that ThriftyDAgger solicits the same number of interventions as HG-DAgger, but requires fewer human and robot actions.

Algorithm	Training Interventions			Train Succ.	Execution Interventions		
	T Ints	T Acts (H)	T Acts (R)		T Ints	T Acts (H)	T Acts (R)
Behavior Cloning	N/A	1381	N/A	N/A	N/A	N/A	N/A
HG-DAgger	31	1682	1199	20/20	6	41	1109
Ours: ThriftyDAgger	27	1728	1153	19/21	6	23	919

This again indicates that ThriftyDAgger is able to learn intervention criteria competitive with human judgment. At execution time, we find that ThriftyDAgger solicits the same number of interventions as HG-DAgger, but requires fewer human and robot actions than HG-DAgger.

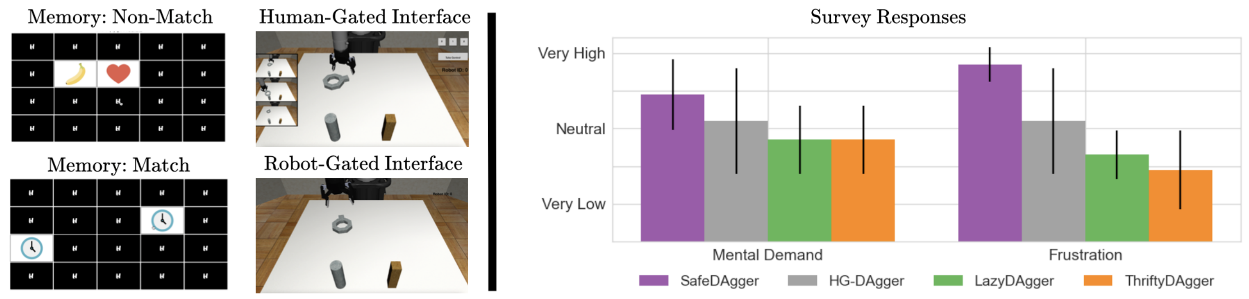


Figure B.2: **User Study Survey Results:** We illustrate the user study interface for the human-gated and robot-gated algorithms (left) and users' survey responses regarding their mental load and frustration (right) for each algorithm. Results suggest that users report similar levels of mental load and frustration for ThriftyDAgger and LazyDAgger, but significantly higher levels of both metrics for HG-DAgger and SafeDAgger. We hypothesize that the sparing and sustained interventions solicited by ThriftyDAgger and LazyDAgger lead to greater user satisfaction and comfort compared to algorithms which force the user to constantly monitor the system (HG-DAgger) or frequently context switch between teleoperation and the distractor task.

B.4 User Study Details

Here we detail the protocol for conducting user studies with ThriftyDAgger and comparisons and discuss qualitative results based on participants' answers to survey questions measuring their mental load and levels of frustration when using each of the algorithms.

B.4.1 User Study Interface

Figure B.2 (left) illustrates the interface used for the user study. The user study is performed with the same peg insertion environment used for simulation experiments, but with 3 robots performing the task in parallel. The base policy is initialized from 30 demos, as in the other simulation experiments. To speed up the task execution for the user study, each action has twice the magnitude as in the peg insertion experiments. Since this results in shorter trajectories that are easier for Behavior Cloning to accomplish, we also inject a small amount of Gaussian noise (covariance matrix $= 0.02\mathcal{N}(0, I)$).

In the human-gated study with HG-DAgger, participants are shown videos of all 3 robots attempting to perform the task in a side pane (Figure B.2, top right of left pane) and are instructed to monitor all of the robots and intervene when they deem it appropriate. In all robot-gated studies, participants are instructed to play the Concentration game until they hear a chime, at which point they are instructed to switch screens to the teleoperation interface. The Concentration game (also called Memory) is illustrated on the left of the left pane in Figure B.2: the objective is to find pairs of cards (all of which are initially face-down) which have matching pictures on their front side. Examples of a non-matching pair and a matching pair are illustrated in Figure B.2.

All robots which require interventions are placed in a FIFO queue, with participants serving intervention requests sequentially until no robot requires intervention. Thus, the participant may be required to provide interventions for multiple robots in succession if multiple robots are currently in the queue. When no robot requires assistance, the teleoperation interface turns black and reports that no robot currently needs help, at which point participants are instructed to return to the Concentration game.

B.4.2 NASA TLX Survey Results

After each participant is subjected to all 4 conditions (SafeDAgger, LazyDAgger, ThriftyDAgger, and HG-DAgger) in a randomized order, we give each participant a NASA TLX survey asking them to rate their mental demand and frustration for each of the conditions on a scale of 1 (very low) - 5 (very high). Results (Figure B.2 right pane) suggest that ThriftyDAgger and LazyDAgger impose less mental demand and make participants feel less frustrated than HG-DAgger and SafeDAgger. During experiments, we found that participants found it cumbersome to keep track of all of the robots simultaneously in HG-DAgger, while the frequent context switches in SafeDAgger made participants frustrated since they were often unable to make much progress in the Concentration Game and felt that the robot repeatedly asked for interventions in very similar states.

B.4.3 Wall Clock Time

We report additional metrics on the wall clock time of each condition in Table B.4. Since all experiments are run for the same 350 time steps, total wall clock time is relatively consistent. However, HG-DAgger takes longer, as it takes more compute to render all three robot views at once. ThriftyDAgger takes less total human time than the baselines, allowing the human to make more progress on independent tasks. Note that other robots in autonomous mode can still make task progress during human intervention. Note also that HG-DAgger requires human attention for the Total Wall Clock Time, as the human must supervise all the robots even if he or she is not actively teleoperating one (as recorded by Human Wall Clock Time).

B.4.4 Detailed Protocol

For the user study, we recruited 10 participants aged 18-37, including members without any knowledge or experience in robotics or AI. All participants are first assigned a randomly selected user ID. Then, participants are instructed to play a 12-card game of Concentration (also known as Memory) (`https://www.helpfulgames.com/subjects/brain-training/memory.html`) in order to learn how to play. Then, users are given practice with both the robot-gated and human-gated teleoperation interfaces. To do this, the operator of the study (one of the authors) performs one episode of the task in the robot-gated interface and briefly

Table B.4: **Wall Clock Time:** We compare the total amount of wall clock time and total amount of human wall clock time averaged over the 10 subjects in the user study. Human Wall Clock Time refers to the amount of time the human spent actively teleoperating a robot, while Total Wall Clock Time measures the amount of time taken by the total experiment. ThriftyDAgger requires the lowest amount of human time, and the total amount of time is relatively consistent. Note that HG-DAgger takes more Total Wall Clock Time as it takes longer to simulate the "bird's eye view" of all 3 robots, and that autonomous robots can still make task progress independently while a human is operating a robot.

Algorithm	Human Wall Clock Time (s)	Total Wall Clock Time (s)
SafeDAgger	448.0 ± 48.1	613.0 ± 33.1
LazyDAgger	415.3 ± 90.3	609.6 ± 49.5
HG-DAgger	532.6 ± 105.2	792.8 ± 68.7
Ours: ThriftyDAgger	$\mathbf{365.4 \pm 88.1}$	625.5 ± 52.3

explains how to control the human-gated interface. Then, participants are instructed to perform one practice episode in the robot-gated teleoperation interface and spend a few minutes exploring the human-gated interface until they are confident in the usage of both interfaces and in how to teleoperate the robots. In the robot-gated experiments, participants are instructed to play Concentration when no robot asks for help, but to immediately switch to helping the robot whenever a robot asks for help. In the human-gated experiment with HG-DAgger, participants are instructed to continuously monitor all of the robots and perform interventions which they believe will maximize the number of successful episodes. During the robot-gated study, participants play the 24-card version of Concentration between robot interventions. If a participant completes the game, new games of Concentration are created until a time budget of robot interactions is hit. Then for each condition, the participant is scored based on (1) the number of times the robot successfully completed the task and (2) the number of total matching pairs the participant found across all games of Concentration.

Appendix C

On-Policy Robot Imitation Learning from a Converging Supervisor

C.1 Static Regret

C.1.1 Proof of Theorem 4.3.1

Recall the standard notion of static regret as defined in Definition 4.3.1:

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

However, we seek to bound

$$\text{Regret}_N^S(\psi_N) = \sum_{i=1}^{N} [l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta^\star}, \psi_N)] \text{ where } \theta^\star = \arg\min_{\theta \in \Theta} \sum_{i=1}^{N} l_i(\pi_\theta, \psi_N) \tag{C.2}$$

as defined in Definition 4.3.2.

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:

$$
\begin{aligned}
\text{Regret}_N^S(\psi_N) &= \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta^\star}, \psi_N)\right] && \text{(C.3)}\\
&= \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta^\star}, \psi_N)\right] - \text{Regret}_N^S((\psi_i)_{i=1}^N) + \text{Regret}_N^S((\psi_i)_{i=1}^N) && \text{(C.4)}\\
&= \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)\right] + \sum_{i=1}^N \left[l_i(\pi_{\theta^*}, \psi_i) - l_i(\pi_{\theta^\star}, \psi_N)\right] \\
&\quad + \text{Regret}_N^S((\psi_i)_{i=1}^N) && \text{(C.5)}\\
&\le \sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)\right] + \sum_{i=1}^N \left[l_i(\pi_{\theta^\star}, \psi_i) - l_i(\pi_{\theta^\star}, \psi_N)\right] \\
&\quad + \text{Regret}_N^S((\psi_i)_{i=1}^N) && \text{(C.6)}
\end{aligned}
$$

Here, inequality C.6 follows from the fact that $\sum_{i=1}^N l_i(\pi_{\theta^*}, \psi_i) \le \sum_{i=1}^N l_i(\pi_{\theta^\star}, \psi_i)$. Now, we can focus on bounding the extra term. Let $h(x, y) = \|x - y\|^2$.

$$
\begin{aligned}
&\sum_{i=1}^N \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i)\right] + \sum_{i=1}^N \left[l_i(\pi_{\theta^\star}, \psi_i) - l_i(\pi_{\theta^\star}, \psi_N)\right] && \text{(C.7)}\\
&= \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)) - h(\pi_{\theta_i}(s_t^i), \psi_i(s_t^i))\right] \\
&\quad + \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T h(\pi_{\theta^\star}(s_t^i), \psi_i(s_t^i)) - h(\pi_{\theta^\star}(s_t^i), \psi_N(s_t^i))\right] && \text{(C.8)}\\
&\le \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T \langle \nabla_\psi h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)), \psi_N(s_t^i) - \psi_i(s_t^i)\rangle\right] \\
&\quad + \sum_{i=1}^N \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T}\sum_{t=1}^T \langle \nabla_\psi h(\pi_{\theta^\star}(s_t^i), \psi_i(s_t^i)), \psi_i(s_t^i) - \psi_N(s_t^i)\rangle\right] && \text{(C.9)}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T}\langle 2(\psi_N(s_t^i)-\pi_{\theta_i}(s_t)), \psi_N(s_t^i)-\psi_i(s_t^i)\rangle\right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T}\langle 2(\psi_i(s_t^i)-\pi_{\theta^\star}(s_t)), \psi_i(s_t^i)-\psi_N(s_t^i)\rangle\right] \qquad \text{(C.10)}\\
&\le \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T} 2\|\psi_N(s_t^i)-\pi_{\theta_i}(s_t)\|\|\psi_N(s_t^i)-\psi_i(s_t^i)\|\right] \\
&\quad + \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T} 2\|\psi_i(s_t^i)-\pi_{\theta^\star}(s_t)\|\|\psi_i(s_t^i)-\psi_N(s_t^i)\|\right] \qquad \text{(C.11)}\\
&\le 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T} \|\psi_N(s_t^i)-\psi_i(s_t^i)\|\right] \qquad \text{(C.12)}
\end{aligned}
$$

Equation C.8 follows from applying the definition of the loss function. Inequality C.9 follows from applying convexity of h in ψ. Equation C.10 follows from evaluating the corresponding gradients. Inequality C.11 follows from Cauchy-Schwarz and inequality C.12 follows from the action space bound. Thus, we have:

$$
\text{Regret}_N^S(\psi_N) \le 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T} \|\psi_N(s_t^i)-\psi_i(s_t^i)\|\right] + \text{Regret}_N^S((\psi_i)_{i=1}^N) \qquad \text{(C.13)}
$$

C.1.2 Proof of Corollary 4.3.1

$$
\forall s \in \mathcal{S},\ \forall N > i, \|\psi_i(s)-\psi_N(s)\| \le f_i \text{ where } \lim_{i\to\infty} f_i = 0 \qquad \text{(C.14)}
$$

implies that

$$
\mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T}\|\psi_i(s_t^i)-\psi_N(s_t^i)\|\right] \le f_i\ \forall N > i \in \mathbb{N} \qquad \text{(C.15)}
$$

This in turn implies that

$$
\sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)}\left[\frac{1}{T}\sum_{t=1}^{T}\|\psi_i(s_t^i)-\psi_N(s_t^i)\|\right] \le \sum_{i=1}^{N} f_i \qquad \text{(C.16)}
$$

Remark: For sublinearity, we really only need inequality C.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 C.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

$$\lim_{n\to\infty} \frac{a_1 + a_2 \ldots a_n}{n} = \lim_{n\to\infty} a_n$$

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

$$\sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] \le \sum_{i=1}^{N} f_i = o(N)$$

Thus, based on the regret bound in Theorem 4.3.1, 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 $\text{Regret}_N^S((\psi_i)_{i=1}^N)$:

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

□

C.2 Dynamic Regret

C.2.1 Proof of Lemma 4.3.1

Recall the standard notion of dynamic regret as defined in Definition 4.3.3:

$$\text{Regret}_N^D((\psi_i)_{i=1}^N) = \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_i) - l_i(\pi_{\theta_i^*}, \psi_i) \right] \text{ where } \theta_i^* = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_i) \tag{C.17}$$

However, we seek to bound

$$\text{Regret}_N^D(\psi_N) = \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i^\star}, \psi_N) \right] \text{ where } \theta_i^\star = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_N) \tag{C.18}$$

as defined in Definition 4.3.4.

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:

$$\text{Regret}_N^D(\psi_N) = \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i^\star}, \psi_N) \right] \tag{C.19}$$

$$= \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i^\star}, \psi_N) \right] - \text{Regret}_N^D((\psi_i)_{i=1}^N) + \text{Regret}_N^D((\psi_i)_{i=1}^N) \tag{C.20}$$

$$= \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i) \right] + \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i^*}, \psi_i) - l_i(\pi_{\theta_i^\star}, \psi_N) \right] + \text{Regret}_N^D((\psi_i)_{i=1}^N) \tag{C.21}$$

$$\leq \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i) \right] + \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i^\star}, \psi_i) - l_i(\pi_{\theta_i^\star}, \psi_N) \right] + \text{Regret}_N^D((\psi_i)_{i=1}^N) \tag{C.22}$$

Here, inequality C.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 $h(x, y) = \|x - y\|^2$.

$$\sum_{i=1}^{N} \left[l_i(\pi_{\theta_i}, \psi_N) - l_i(\pi_{\theta_i}, \psi_i) \right] + \sum_{i=1}^{N} \left[l_i(\pi_{\theta_i^\star}, \psi_i) - l_i(\pi_{\theta_i^\star}, \psi_N) \right] \tag{C.23}$$

$$= \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)) - h(\pi_{\theta_i}(s_t^i), \psi_i(s_t^i)) \right] + \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} h(\pi_{\theta_i^\star}(s_t^i), \psi_i(s_t^i)) - h(\pi_{\theta_i^\star}(s_t^i), \psi_N(s_t^i)) \right] \tag{C.24}$$

$$\leq \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle \nabla_\psi h(\pi_{\theta_i}(s_t^i), \psi_N(s_t^i)), \psi_N(s_t^i) - \psi_i(s_t^i) \rangle \right] + \sum_{i=1}^{N} \mathbb{E}_{\tau \sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle \nabla_\psi h(\pi_{\theta_i^\star}(s_t^i), \psi_i(s_t^i)), \psi_i(s_t^i) - \psi_N(s_t^i) \rangle \right] \tag{C.25}$$

$$= \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle 2(\psi_N(s_t^i) - \pi_{\theta_i}(s_t)), \psi_N(s_t^i) - \psi_i(s_t^i) \rangle \right]$$
$$+ \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \langle 2(\psi_i(s_t^i) - \pi_{\theta_i^\star}(s_t)), \psi_i(s_t^i) - \psi_N(s_t^i) \rangle \right] \tag{C.26}$$

$$\leq \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2\|\psi_N(s_t^i) - \pi_{\theta_i}(s_t)\| \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right]$$
$$+ \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} 2\|\psi_i(s_t^i) - \pi_{\theta_i^\star}(s_t)\| \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] \tag{C.27}$$

$$\leq 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] \tag{C.28}$$

The steps of this proof follow as in the proof of the static regret reduction. Equation C.24 follows from applying the definition of the loss function. Inequality C.25 follows from applying convexity of h in ψ. Equation C.26 follows from evaluating the corresponding gradients. Inequality C.27 follows from Cauchy-Schwarz and inequality C.28 follows from the action space bound. Combining this bound with C.22, we have our desired result:

$$\text{Regret}_N^D(\psi_N) \leq 4\delta \sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_N(s_t^i) - \psi_i(s_t^i)\| \right] + \text{Regret}_N^D((\psi_i)_{i=1}^N) \tag{C.29}$$

□

C.2.2 Proof of Corollary 4.3.2

By Corollary 4.3.1,

$$\sum_{i=1}^{N} \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \left[\frac{1}{T} \sum_{t=1}^{T} \|\psi_i(s_t^i) - \psi_N(s_t^i)\| \right] = o(N)$$

which implies that

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

□

C.2.3 Predictability of Online Learning Problems

Next, we establish that the online learning problem defined by the losses defined in Section 4.2 is an (α, β)-predictable online learning problem as defined in Cheng et al. [97]. 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. [97] shows that for (α, β)-predictable problems, sublinear dynamic regret can be achieved if $\alpha > \beta$. Furthermore, Theorem 3 in Cheng et al. [97] shows that if α is sufficiently large and β sufficiently small, then sublinear dynamic regret can be achieved by online gradient descent.

Lemma C.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 4.3.1 holds, then $l_i(\pi_\theta, \psi)$ satisfies:*

$$\|\nabla_\theta l_{i+1}(\pi_\theta, \psi_{i+1}) - \nabla_\theta l_i(\pi_\theta, \psi_i)\| \leq 4G\eta \sup_{a\in\mathcal{A}} \|a\| \|\theta_{i+1} - \theta_i\| + \zeta_i \text{ where } \sum_{i=1}^N \zeta_i = o(N)$$

Proof of Lemma C.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. [97].

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

$$\begin{aligned}
\nabla_\theta l_i(\pi_\theta, \psi) &= \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \frac{1}{T}\sum_{t=1}^T \nabla_\theta \|\psi(s_t) - \pi_\theta(s_t)\|^2 && \text{(C.30)}\\
&= \mathbb{E}_{\tau\sim p(\tau|\theta_i)} \nabla_\theta J_\tau(\pi_\theta, \psi) && \text{(C.31)}\\
&= \int p(\tau|\theta_i) \nabla_\theta J_\tau(\pi_\theta, \psi) d\tau && \text{(C.32)}\\
\nabla_\theta J_\tau(\pi_\theta, \psi) &= \frac{1}{T}\sum_{s_t\in\tau} 2\nabla_\theta \pi_\theta(s_t)^T (\pi_\theta(s_t) - \psi(s_t)) && \text{(C.33)}\\
&= \frac{2}{T} \nabla_\theta \pi_\theta(\tau)^T (\pi_\theta(\tau) - \psi(\tau)) && \text{(C.34)}
\end{aligned}$$

where

$$\psi(\tau) = \begin{bmatrix} \psi(s_0) \\ \vdots \\ \psi(s_T) \end{bmatrix},\ \pi_\theta(\tau) = \begin{bmatrix} \pi_\theta(s_0) \\ \vdots \\ \pi_\theta(s_T) \end{bmatrix},\ \nabla_\theta \pi_\theta(\tau) = \begin{bmatrix} \nabla_\theta \pi_\theta(s_0) \\ \vdots \\ \nabla_\theta \pi_\theta(s_T) \end{bmatrix} \quad \text{(C.35)}$$

Taking the difference of the above loss gradients, we obtain:

$$
\begin{aligned}
&\|\nabla_\theta l_{i+1}(\pi_\theta, \psi_{i+1}) - \nabla_\theta l_i(\pi_\theta, \psi_i)\| && \text{(C.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\| && \text{(C.37)}\\
&\le \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 && \text{(C.38)}\\
&= \int \Big\| \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)) \\
&\quad + \frac{2}{T}\nabla_\theta \pi_\theta(\tau)^T(p(\tau|\theta_{i+1})\pi_\theta(\tau) - p(\tau|\theta_i)\pi_\theta(\tau)) \Big\| d\tau && \text{(C.39)}\\
&\le \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 \\
&\quad + \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 && \text{(C.40)}\\
&\le \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 \\
&\quad + 2G \sup_{a\in\mathcal{A}} \|a\| \int |p(\tau|\theta_{i+1}) - p(\tau|\theta_i)| d\tau && \text{(C.41)}\\
&\le \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\| && \text{(C.42)}\\
&\le \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\| && \text{(C.43)}\\
&= \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 \\
&\quad + 2G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| && \text{(C.44)}
\end{aligned}
$$

$$
\begin{aligned}
&\le \frac{2}{T}G \int \|p(\tau|\theta_i)(\psi_i(\tau) - \psi_{i+1}(\tau))\| + \|(p(\tau|\theta_i) - p(\tau|\theta_{i+1}))\psi_{i+1}(\tau)\| d\tau \\
&\quad + 2G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| && \text{(C.45)}\\
&\le \frac{2}{T}G \int p(\tau|\theta_i)\|\psi_i(\tau) - \psi_{i+1}(\tau)\| d\tau + 4G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| && \text{(C.46)}\\
&\le 2Gf_i \int p(\tau|\theta_i)d\tau + 4G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| && \text{(C.47)}\\
&\le 2Gf_i + 4G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| && \text{(C.48)}\\
&= 4G\eta \sup_{a\in\mathcal{A}} \|a\|\|\theta_{i+1} - \theta_i\| + \zeta_i && \text{(C.49)}
\end{aligned}
$$

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 Corollary 4.3.1. Equation C.37 follows from applying definitions. Equation C.38 follows from the triangle inequality. Equation C.39 follows from substitution of the loss gradients. Inequality C.40 follows from the triangle inequality and factoring out common terms. Inequality C.41 follows from subadditivity, the policy Jacobian and action space bound. Inequality C.42 follows from Assumption 4.3.1. Equation C.43 follows from subadditivity of the operator norm and the policy Jacobian bound. Equation C.45 follows from the triangle inequality, and equation C.46 follows from the triangle inequality and Assumption 4.3.1. Equations C.47 and C.49 follow from the convergence assumption of the supervisor and the triangle inequality. □

Lemma C.2.2. *Assumption 4.2.2 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 Lemma C.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

$$\forall \theta, \theta_i \in \Theta, \forall \psi, \|\nabla_\theta l_i(\pi_\theta, \psi)\| \leq 2G\delta \quad \square$$

C.2.4 Proof of Lemma 4.3.2

From Lemma C.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. [97]. Thus, by Proposition 12 from Cheng et al. [97], 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 [97]: $\theta_{i+1} = \arg\min_{\theta\in\Theta} l_i(\pi_\theta, \psi_i)$.

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. [97]:

$$\|\theta_i^* - \theta_i\| = \|\theta_i^* - \theta_{i-1}^*\| \leq \lambda\|\theta_i - \theta_{i-1}\| + \frac{\zeta_i}{\alpha} \leq \lambda^i\|\theta_1 - \theta_0\| + \sum_{j=1}^{i} \lambda^{i-j}\xi_j$$

where the first inequality follows from Proposition 1 of Lee et al. [93] 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 Lemma 4.3.2, 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 Lemma C.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. [97] with $\frac{4G\eta\sup_{a\in\mathcal{A}}\|a\|}{\alpha} > \frac{\alpha}{2\gamma}$ if the losses are γ-smooth in θ. □

C.2.5 Proof of Theorem 4.3.2

The proof follows immediately from combining the result of Corollary 4.3.2 and Lemma 4.3.2. □

C.3 Training Details

C.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.

C.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 [346]. 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. [86] contains further details on training PETS.

C.3.3 SAC

We use the rlkit implementation [347] 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.

C.3.4 TD3

We use the rlkit implementation [347] 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.

C.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 [348] 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.

C.4 Experimental Details

C.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. [86]. Both are implemented as 7-DOF torque control tasks. For all tasks, we plot the sum of rewards for each training episode.

C.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 [142]. 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.

Appendix D

Safe Learning MPC for Stochastic Nonlinear Dynamical Systems with Adjustable Boundary Conditions

D.1 Proofs of Controller Properties

Proof of Lemma 5.5.1 We proceed by induction. By assumption 5.3.1, $J^0_{0\to H}(x^j_0) < \infty$. By the definition of $V^{\pi^j}_{\mathcal{G}}$ and $\mathcal{F}^j_{\mathcal{G}}$, $J^j_{0\to H}(x^j_0) < \infty$. 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:

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

$$= C(x^j_t, \pi^{*,j}_{t|t}(x^j_t)) + \mathbb{E}_{w^j_{t:t+H-1}}\left[\sum_{k=1}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H|t})\right] \tag{D.2}$$

$$= C(x^j_t, \pi^{*,j}_{t|t}(x^j_t))$$
$$+ \underset{w^j_{t:t+H}}{\mathbb{E}}\left[\sum_{k=1}^{H-1} C(x^j_{t+k|t}, \pi^{*,j}_{t+k|t}(x^j_{t+k|t})) + C(x^j_{t+H|t}, \pi^l(x^j_{t+H|t})) \right. \tag{D.3}$$
$$\left. + V^{\pi^{j-1}}_{\mathcal{G}}(x^j_{t+H+1|t})\right],\ l \in [j-1] \tag{D.4}$$

$$\begin{aligned}
&\geq C(x_t^j, \pi_{t|t}^{*,j}(x_t^j)) \\
&+ \mathop{\mathbb{E}}_{w_t^j}\Bigg[\min_{\pi_{t+1:t+H|t+1}} \mathop{\mathbb{E}}_{w_{t+1:t+H}^j}\Bigg[\sum_{k=1}^{H-1} C(x_{t+k|t+1}^j, \pi_{t+k|t+1}(x_{t+k|t+1}^j)) \\
&+ C(x_{t+H|t+1}^j, \pi_{t+H|t+1}(x_{t+H|t+1}^j)) \\
&+ V_{\mathcal{G}}^{\pi^{j-1}}(x_{t+H+1|t+1}^j)\Bigg]\Bigg]
\end{aligned} \tag{D.5}$$

$$= C(x_t^j, \pi^j(x_t^j)) + \mathop{\mathbb{E}}_{w_t^j}\left[J_{t+1\to t+H+1}^j(x_{t+1}^j)|x_t^j\right] \tag{D.6}$$

Equation D.1 follows from the definition in 5.10, equation D.3 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 $x_{t+H|t}^j$ 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 D.5 conditions on the random variable x_{t+1}^j, and the outer expectation integrates it out. The inequality in D.5 follows from the fact that $[\pi_{t+1|t}^{*,j}, \ldots, \pi_{t+H-1|t}^{*,j}, \pi^{j-1}]$ is a possible solution to (D.5). Equation D.6 follows from the definition in equation 5.10.

We have shown that $J_{t\to t+H}^j(x_t^j) < \infty \implies \mathop{\mathbb{E}}_{w_t^j}\left[J_{t+1\to t+H+1}^j(x_{t+1|t}^j)\right] < \infty$. So:

$$\mathop{\mathbb{E}}_{w_{0:t-1}^j}\left[J_{t\to t+H}^j(x_t^j)\right] < \infty \implies \mathop{\mathbb{E}}_{w_{0:t-1}^j}\left[\mathop{\mathbb{E}}_{w_t^j}\left[J_{t+1\to t+H+1}^j(x_{t+1|t}^j)\right]\right] \tag{D.7}$$

$$= \mathop{\mathbb{E}}_{w_{0:t}^j}\left[J_{t+1\to t+H+1}^j(x_{t+1}^j)\right] < \infty \tag{D.8}$$

By induction, $\mathop{\mathbb{E}}_{w_{0:t-1}^j}[J_{t\to t+H}^j(x_t^j)] < \infty\ \forall t \in \mathbb{N}$. Therefore, the controller is feasible at iteration j. □

Proof of Lemma 5.5.2 By Lemma 5.5.1 and Assumption 5.2.1, $\forall L \in \mathbb{N}$,

$$\mathop{\mathbb{E}}_{w_{1:L-1}^j}\left[\sum_{k=0}^{L-1} C(x_k^j, \pi^j(x_k^j)) + J_{L\to L+H}^j(x_L^j)\right] \leq J_{0\to H}^j(x_0^j) \tag{D.9}$$

$$\implies \mathop{\mathbb{E}}_{w_{1:L-1}^j}\left[J_{L\to L+H}^j(x_L^j)\right] \leq J_{0\to H}^j(x_0^j) - \mathop{\mathbb{E}}_{w_{1:L-1}^j}\left[\sum_{k=0}^{L-1} C(x_k^j, \pi^j(x_k^j))\right] \tag{D.10}$$

$$\leq J_{0\to H}^j(x_0^j) - \epsilon \sum_{k=0}^{L-1} P(x_k^j \notin \mathcal{G}) \tag{D.11}$$

Line D.11 follows from rearranging D.9 and applying assumption 5.2.1. Because $\mathcal{G}$ is robust control invariant by assumption 5.2.2, $x_t \in \mathcal{G} \implies x_{t+k} \in \mathcal{G}\ \forall k \geq 0$. Now, assume $\lim_{k\to\infty} P(x_k^j \notin \mathcal{G})$ does not exist or is nonzero. This implies that $P(x_k^j \notin \mathcal{G}) \geq \delta > 0$ infinitely many times. By the Archimedean principle, the RHS of D.11 can be driven arbitrarily negative, which is impossible. By contradiction, $\lim_{k\to\infty} P(x_k^j \notin \mathcal{G}) = 0$. □

Proof of Theorem 5.5.1 Let $j \in \mathbb{N}$

$$J^j_{0\to H}(x_0) \geq C(x_0,u_0) + \mathop{\mathbb{E}}_{w_0^j}\left[J^j_{1\to H+1}(x_1^j)\right] \tag{D.12}$$

$$\geq \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty}\mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x_t^j)\right] \tag{D.13}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty}\mathop{\mathbb{E}}_{\mathbb{1}\{x_t^j\notin\mathcal{G}\}}\left[\mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x_t)|\mathbb{1}\{x_t^j \notin \mathcal{G}\}\right]\right] \tag{D.14}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty}\mathop{\mathbb{E}}_{w^j_{0:t-1}}\left[J^j_{t\to t+H}(x_t^j)|x_t^j \notin \mathcal{G}\right] P(x_t^j \notin \mathcal{G}) \tag{D.15}$$

$$\geq \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] + \lim_{t\to\infty} \epsilon P(x_t^j \notin \mathcal{G}) \tag{D.16}$$

$$= \mathop{\mathbb{E}}_{w^j}\left[\sum_{t=0}^{\infty} C(x_t^j, \pi^j(x_t^j))\right] = J^{\pi^j}(x_0) \tag{D.17}$$

Equations D.12 and D.13 follow from repeated application of Lemma 5.5.1 (D.6). Equation D.14 follows from iterated expectation, equation D.15 follows from the cost function assumption 5.2.1. Equation D.16 follows again from assumption 5.2.1 (incur a cost of at least ϵ for not being at the goal at time t). Then, Equation D.17 follows from Lemma 5.5.2. Using the above inequality with the definition of $J^{\pi^j}(x_0)$,

$$J^j_{0\to H}(x_0) \geq J^{\pi^j}(x_0) = \mathop{\mathbb{E}}_{w^j_{0:H-1}}\left[\sum_{t=0}^{H-1} C(x_t^j, \pi^j(x_t)) + V_{\mathcal{G}}^{\pi^j}(x_H^j)\right] \tag{D.18}$$

$$\geq \mathop{\mathbb{E}}_{w^j_{0:H-1}}\left[\sum_{t=0}^{H-1} C(x_t^j, \pi^{*,j}_{t|0}(x_{t|0})) + V_{\mathcal{G}}^{\pi^j}(x_{H|0})\right] = J^{j+1}_{0\to H}(x_0) \tag{D.19}$$

$$\geq J^{\pi^{j+1}}(x_0) \tag{D.20}$$

Equation D.18 follows from equation D.17, equation D.19 follows from taking the minimum over all possible H-length sequences of policies in the policy class Π. Equation D.20 follows from equation D.17. 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. □

Proof of Lemma 5.5.3 The proof is identical to [109]. Because $\mathcal{SS}_{\mathcal{G}}^j$ is an increasing sequence of sets, $\mathcal{F}_{\mathcal{G}}^j$ is also an increasing sequence of sets by definition. □

D.2 Adjustable Boundary Condition LMPC Implementation Details

D.2.1 Solving the MPC Problem

As in [106], 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 [86]. 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_S^j from $\bigcup_{k=0}^{j} \tilde{\mathcal{SS}}^k$, we sort states in the safeset under $C_E^j(x)$ and use this to choose x_S^j close to x^* under $C_E^j(x)$. Note that this choice does not impact any of the theoretical guarantees.

D.2.2 Value Function

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 $L^{\pi^j}(x)$.

D.2.3 Start State Expansion

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.

D.3 Experiment Specific Parameters

D.3.1 Pointmass Navigation

Environment Details: 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 controller. The task horizon is set to $T = 50$.

Task Controller MPC Parameters: For the single start, single goal set case we use popsize = 400, num_elites = 40, cem_iters = 5, and plan_hor = 15. For all start state expansion experiments, we utilize the same popsize, num_elites, and cem_iters but utilize plan_hor = 20. For experiments we utilize $\alpha = 2$ for the kernel width parameter for density model $\rho_\alpha^{\mathcal{G}}$.

Start State Expansion Parameters: We utilize $H' = H - 5$ for all experiments (trajectory optimization horizon for exploration policy).

SAVED Baseline Experimental Parameters: We supply SAVED with 100 demonstrations generated by the same demonstration policy as for ABC-LMPC. We utilize $\alpha = 3$ and utilize the implementation from [106]. Both the value function and dynamics for SAVED are represented with a probabilistic ensemble of 5 neural networks with 3 hidden layers of 500 hidden units each. We use swish activations, and update weights using the Adam Optimizer with learning rate 0.001.

D.3.2 7-Link Reacher Arm

Environment Details: 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 [136].

Task Controller MPC Parameters: For the single start, single goal set case we use popsize = 400, num_elites = 40, cem_iters = 5, and plan_hor = 15. For all start state expansion experiments, we utilize the same popsize, num_elites, and cem_iters but utilize plan_hor = 20. For experiments we utilize $\alpha = 0.5$ for the kernel width parameter for density model $\rho_\alpha^\mathcal{G}$.

Start State Expansion Parameters: We utilize $H' = H - 5$ for all experiments (trajectory optimization horizon for exploration policy).

SAVED Baseline Experimental Parameters: We supply SAVED with 100 demonstrations generated by the same demonstration policy as for ABC-LMPC. We utilize $\alpha = 0.5$ and utilize the implementation from [106]. Both the value function and dynamics for SAVED are represented with a probabilistic ensemble of 5 neural networks with 3 hidden layers of 500 hidden units each. We use swish activations, and update weights using the Adam Optimizer with learning rate 0.001.

D.3.3 Inverted Pendulum

Environment Details: We use $\sigma = 0.5$ 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 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.

Task Controller MPC Parameters: 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 $\rho_\alpha^\mathcal{G}$.

Start State Expansion Parameters: We utilize $H' = H$ for all experiments (trajectory optimization horizon for exploration policy).

D.4 Controller Domain Expansion Strategy

Here we discuss how the controller 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:

$$\begin{aligned}
\pi^j_{E,0:H'-1} = \underset{\pi_{0:H'-1}\in\Pi^{H'}}{\operatorname{argmin}} \quad & \underset{w^j_{0:H'-2}}{\mathbb{E}}\left[\sum_{i=0}^{H'-1} C^j_E(x^j_i, \pi_i(x^j_i))\right] \\
\text{s.t.} \quad & x^j_{i+1} = f(x^j_i, \pi_i(x^j_i), w_i) \;\forall i \in \{0, \dots, H'-1\} \\
& x^j_{H'} \in \bigcup_{k=0}^{j-1} \mathcal{SS}^k_{\mathcal{G}},\; \forall w_{0:H'-2} \in \mathcal{W}^{H'-1} \\
& x^j_{0:H'} \in \mathcal{X}^{H'+1},\; \forall w_{0:H'-2} \in \mathcal{W}^{H'-1}
\end{aligned} \tag{D.21}$$

For all $x^j_S \in \mathcal{SS}^{j-1}_{\mathcal{G}}$, the states $\bigcup_{k=0}^{H'} \mathcal{R}_k^{\pi^j_{E,0:H'-1}}(x^j_S) \cup \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{R}_{H'}^{\pi^j_{E,0:H'-1}}(x^j_S))$ 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 $\mathcal{SS}^{\pi^j}_{\mathcal{G}} = \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{R}_{H'}^{\pi^j_{E,0:H'-1}}(\mathcal{SS}^{j-1}_{\mathcal{G}})) \cup \bigcup_{k=1}^{\infty} \mathcal{R}_k^{\pi^j}(\mathcal{SS}^{j-1}_{\mathcal{G}})$, we redefine $L^{\pi^j}_{\mathcal{G}}$ as:

$$L^{\pi^j}_{\mathcal{G}}(x) = \begin{cases} \underset{w}{\mathbb{E}}\left[C(x, \pi^j(x)) + L^{\pi^j}_{\mathcal{G}}(f(x, \pi^j(x), w))\right] & x \in \mathcal{SS}^{\pi^j}_{\mathcal{G}} \setminus \mathcal{G} \\ \underset{w}{\mathbb{E}}\left[C(x, \pi^j_{E,0:H'-1}(x)) + L^{\pi^j}_{\mathcal{G}}(f(x, \pi^j_{E,0:H'-1}(x), w))\right] & x \in \mathcal{SS}^j_{\mathcal{G}} \setminus \mathcal{SS}^{\pi^j}_{\mathcal{G}} \\ 0 & x \in \mathcal{G} \\ +\infty & x \notin \mathcal{SS}^j_{\mathcal{G}} \end{cases} \tag{D.22}$$

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 $\mathcal{F}^j_{\mathcal{G}}$ while still maintaining its theoretical properties.

Appendix E

SAVED: Safe Deep Model-Based RL for Sparse Cost Robotic Tasks

E.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.

E.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. [86]. To plan over the dynamics, the TS-∞ trajectory sampling method from [86] 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 [346] of PETS in [86].

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 work. 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, Co-Reyes, and Levine [349], 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 E.1.

E.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.

E.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.

E.1.4 SACfD

We use the rlkit implementation [350] 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 E.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.

E.1.5 OEFD

We use the implementation of OEFD provided by Jangir [351] 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 $k = 4$ [131]. 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 E.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.

E.2 Simulated Experiments Additional Results

In Figure E.1, we illustrate the mechanism by which SAVED iteratively improves upon suboptimal demonstrations on navigation task 2 by planning into an expanding safe set.

In Figure E.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 corresponding chapter.

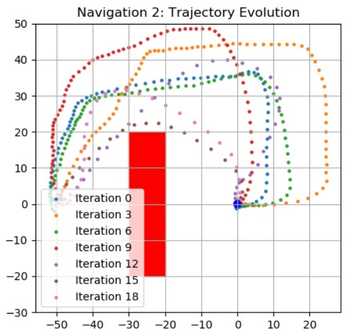


Figure E.1: **Navigation Task 2 Trajectory Evolution:** SAVED rapidly improves upon demonstration trajectories by constraining its exploration to regions of relative certainty and high cost. By iteration 15, SAVED is able to find a safe but efficient trajectory to the goal at the origin.

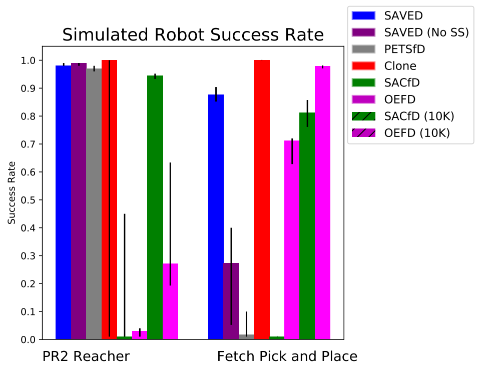


Figure E.2: **Simulated Robot Experiments Success Rate:** SAVED has comparable success rate to Clone, PETSfD, and SAVED (No SS) on the reacher task in the first 100 iterations. For the pick and place task, SAVED outperforms all baselines in the first 250 iterations except for Clone, which does not improve upon demonstration performance.

E.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.

E.3.1 Figure-8

In addition to the knot-tying task discussed in the main paper, 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 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 E.3 and E.4 below. In Figure E.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 E.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 E.4.

In Figure E.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-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.

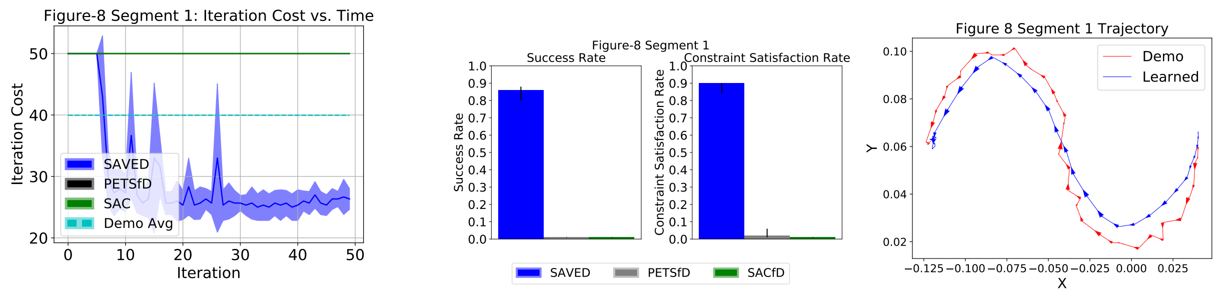


Figure E.3: **Figure-8: Training Performance:** After just 10 iterations, SAVED consistently succeeds and converges to an iteration cost of 26, faster than demos which took an average of 40 steps. Neither baseline ever completes the task in the first 50 iterations; **Trajectories:** Demo trajectories satisfy constraints, but are noisy and inefficient. SAVED learns to speed up with only occasional constraint violations and stabilizes in the goal set.

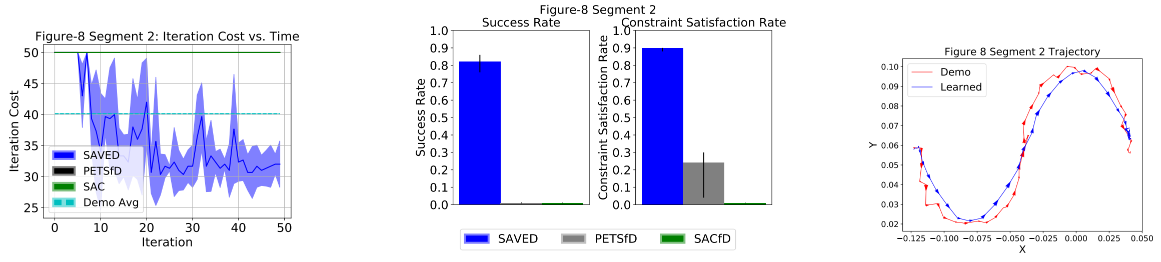


Figure E.4: **Figure-8: Training Performance:** After 10 iterations, the agent consistently completes the task and converges to an iteration cost of around 32, faster than demos which took an average of 40 steps. Neither baseline ever completed the task in the first 50 iterations; **Trajectories:** Demo 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.

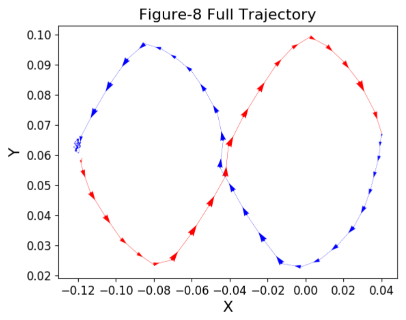


Figure E.5: **Full Figure-8 trajectory:** We show the full figure-8 trajectory, obtained by evaluating learned policies for the first and second figure-8 segments in succession. Even when segmenting the task, the agent can smoothly interpolate between the segments, successfully navigating the uncertainty in the transition at execution-time and stabilizing in the goal set.

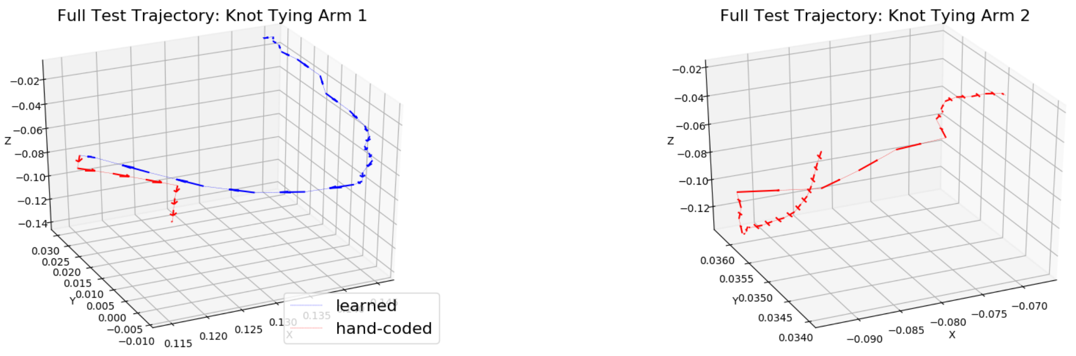


Figure E.6: **Knot-Tying Full Trajectories: (a) Arm 1 trajectory:** We see that the learned part of the arm 1 trajectory is significantly smoothed compared to the demonstrations at execution-time as well, consistent with the training results. Then, in the hand-coded portion of the trajectory, arm 1 is 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.

E.3.2 Knot-Tying

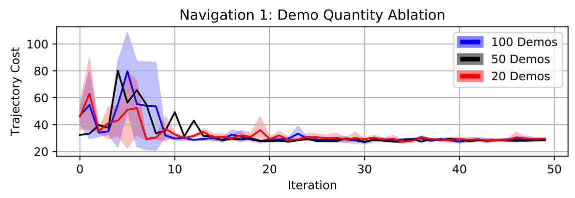
In Figure E.6, we show the full trajectory for both arms for the surgical knot-tying task. We see that the learned policy for arm 1 smoothly navigates around arm 2, after which arm 1 is manually moved down along with arm 2, which grasps the thread and pulls it up through the resulting loop in the thread, completing the knot.

E.4 Ablations

E.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 E.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 E.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 E.7. We note that again, SAVED is relatively robust to varying demonstration quality, achieving similar performance even for very slow demonstrations



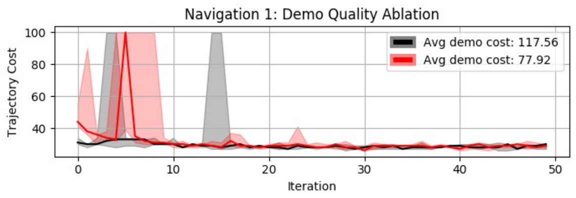


Figure E.7: **SAVED Ablations on Navigation Task 1: Number of Demonstrations:** SAVED is able to consistently complete the task with just both demo qualities considered without significant performance decay. The 100 demonstrations provided in this task have average trajectory cost of 117.56 (black) and 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.

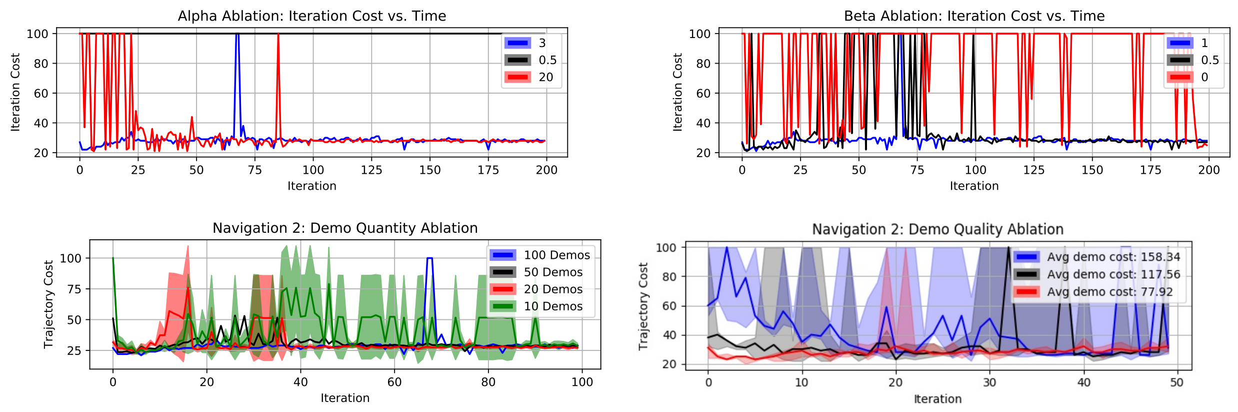


Figure E.8: **SAVED Ablations on Navigation Task 2: Kernel width α:** We see that α must be chosen to be high enough such that SAVED is able to explore enough to find the goal set, but not so high that SAVED starts to explore unsafe regions of the state space; **Chance constraint parameter β:** Decreasing 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.

E.4.2 Model-Free

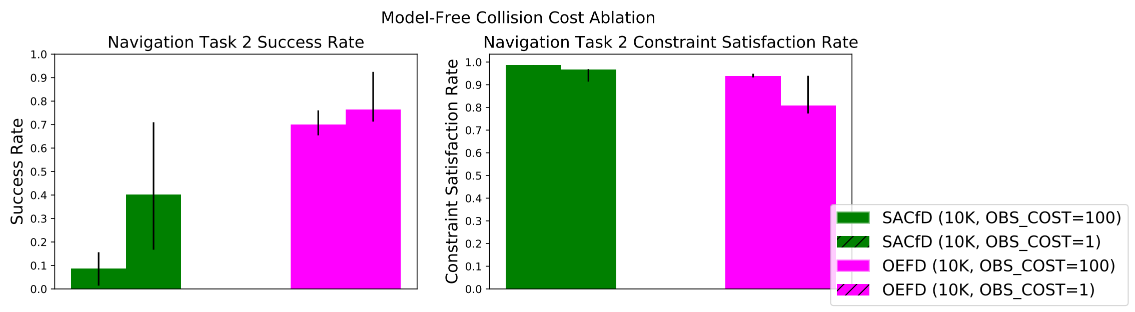
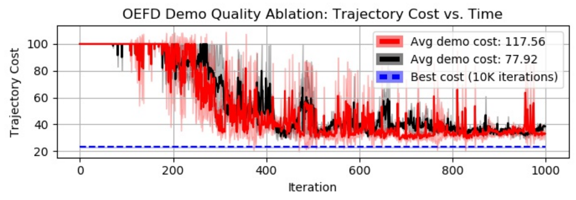


Figure E.9: A high cost for constraint violations results in conservative behavior that learns to avoid the obstacle, but also makes it take longer to learn to perform the task. Setting the cost low results in riskier behavior that more often achieves task success.

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 E.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 E.10), and in both cases, OEFD takes much longer than SAVED to start performing the task successfully (Figure E.7). We also perform the same study on the model-free RL baseline algorithm Soft Actor Critic from Demonstrations (SACfD) [103]. 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 E.10) while SAVED converges in less than 10 iterations (Figure E.7). We also ablate demo quantity for SACfD on navigation task 1 in Figure E.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 E.7).



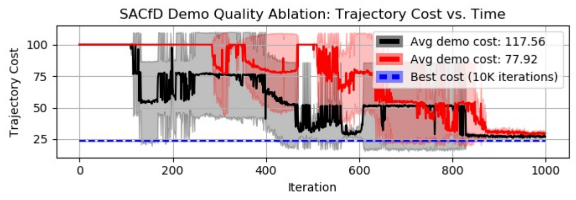


Figure E.10: **SAVED Model-Free RL Demo Quality Ablations on Navigation Task 1: OEFD:** We see that the baseline OEFD has similar performance across demonstration qualities. OEFD takes hundreds of iterations to start performing the task successfully, while SAVED converges less than 10 iterations; **SACfD:** 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.

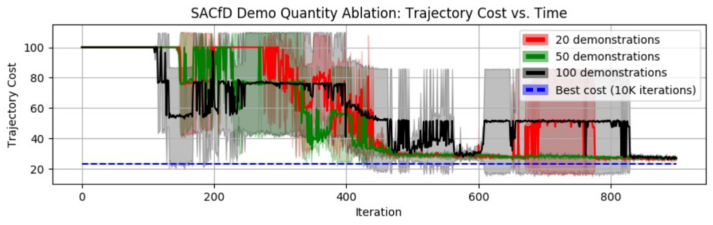


Figure E.11: **SACfD Demo Quantity Ablation on Navigation Task 1:** We study the effect of varying demonstration numbers on the model free RL baseline algorithm SACfD [103]. We see that the baseline SACfD has high variance across all demonstration quantities, and takes roughly similar time to converge 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 E.10), the SACfD runs all converge close to this value.

Appendix F

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

In Appendices F.1 and F.2 we discuss algorithmic details and implementation/hyperparameter details respectively for LS^3 and all comparisons. We then provide full details regarding each of the experimental domains and how data is collected in these domains in Appendix F.3. In Appendix F.4, we present an additional experiment studying the task success rate of LS^3 and comparisons evolves as training progresses. Finally, in Appendix F.5 we perform sensitivity experiments and ablations.

F.1 Algorithm Details

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

F.1.1 Latent Space Safe Sets (LS^3)

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

Variational Autoencoders: We scale all image inputs to a size of $(64, 64, 3)$ before feeding them to the β-VAE, which uses a convolutional neural network for f_{enc} and a transpose convolutional neural network for f_{dec}. We use the encoder and decoder from Hafner et al. [163], 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

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

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

Probabilistic Dynamics: As in Chua et al. [191] 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:

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

When using f_{dyn} for planning, we use the TS-1 method from Chua et al. [191].

Value Functions: As discussed in Section 7.3.3, 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 [106], 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:

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

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

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

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

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

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 [103, 352, 106], 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 7.2. We build on the implementation of the cross entropy method provided in [346], 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 7.2 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 7.2. 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. [191] for more details on the cross entropy method as applied to planning over neural network dynamics models.

As mentioned in Section 7.3.4, 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 $\delta_{\mathbb{S}} = 0.8$.

F.1.2 Soft Actor-Critic from Demonstrations (SACfD)

We utilize the implementation of the Soft Actor Critic algorithm from [350] 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 [350] 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.

F.1.3 Soft Actor-Critic from Demonstrations with Learned Recovery Zones (SACfD+RRL)

We build on the implementation of the Recovery RL algorithm [46] provided in [353]. 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: $(\gamma_{risk}, \epsilon_{risk})$. For each of the simulation environments, we evaluated SACfD+RRL across 3-4 $(\gamma_{risk}, \epsilon_{risk})$ settings and reported results from the highest performing run. Accordingly, for the navigation environment, we use: $(\gamma_{risk} = 0.95, \epsilon_{risk} = 0.8)$. For the reacher environment, we use $(\gamma_{risk} = 0.55, \epsilon_{risk} = 0.7)$, and we use $(\gamma_{risk} = 0.75, \epsilon_{risk} = 0.7)$ for the sequential pushing environment. For the cable routing environment, we use $(\gamma_{risk} = 0.55, \epsilon_{risk} = 0.7)$.

F.1.4 Advantage Weighted Actor-Critic (AWAC)

To provide a comparison to state of the art offline reinforcement learning algorithms, we evaluate AWAC [194] on the experimental domains in this work. We use the implementation of AWAC from [354]. 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 [354] for all other hyperparameters.

F.2 LS^3 Implementation Details

In Table F.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 $n_{particles}$, which is used to provide an estimated probability of constraint satisfaction and expected rewards. For the cross entropy method, we sample $n_{candidate}$ action sequences at each iteration, take the best n_{elite} action

Table F.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

sequences and then refit the sampling distribution. This process iterates n_{cem_iters} times. We also report the latent space dimension d, whether frame stacking is used as input, training batch size, and discount factor γ. Finally, we present values of the safe set bellman coefficient $\gamma_{\mathbb{S}}$. For all domains, we scale RGB observations to a size of $(64, 64, 3)$. For all modules we use the Adam optimizer with a learning rate of 1×10^{-4}, except for dynamics which use a learning rate of 1×10^{-3}.

F.3 Experimental Domain Details

F.3.1 Navigation

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

F.3.2 Reacher

The reacher domain is built on the reacher domain provided in the DeepMind Control Suite from [196]. 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.

F.3.3 Sequential Pushing

This sequential pushing environment is implemented in MuJoCo [61], 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.

F.3.4 Physical Cable Routing

This task starts with the robot grasping one endpoint of the red cable, and it can make $(\Delta x, \Delta y)$ motions with its end effector. The goal is to guide the red cable to intersect with the green goal set while avoiding the blue obstacle. The ground-truth goal and obstacle checks are performed with color masking. LS^3 and all baselines are provided with a segmentation mask of the cable as input. The demonstrator generates trajectories $\mathcal{D}_{success}$ 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 $\mathcal{D}_{rand}$ 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 $|\mathcal{D}_{\text{VAE}}| > 100000$.

F.4 Additional Results

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

F.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-ls3-ls3 F.2 and F.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 (Figure F.4) 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.

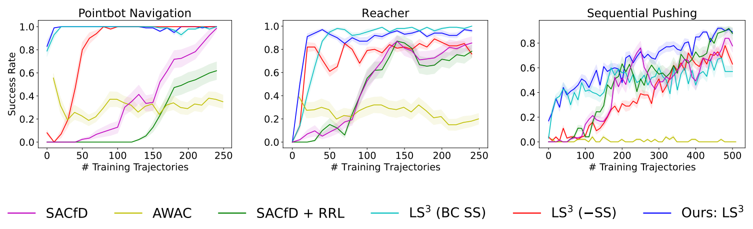


Figure F.1: **Task Success Rate:** Learning curves showing mean and standard error of task success rate of checkpointed policies over 10 random seeds (and 10 rollouts per seed). We see that LS^3 has a much higher task success rate than comparisons early on, and maintains a success rate at least as high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

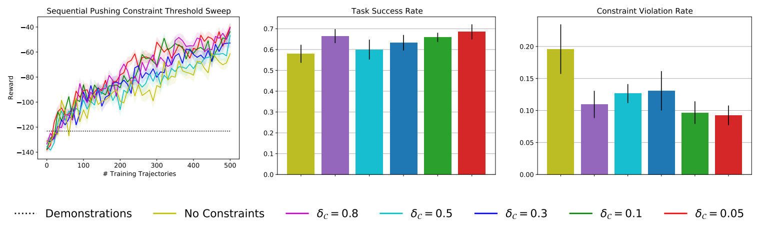


Figure F.2: **Hyperparameter Sweep for LS^3 Constraint Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathcal{C}}$ on the sequential pushing environment. As expected, we see that without avoiding latent space obstacles (No Constraints)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

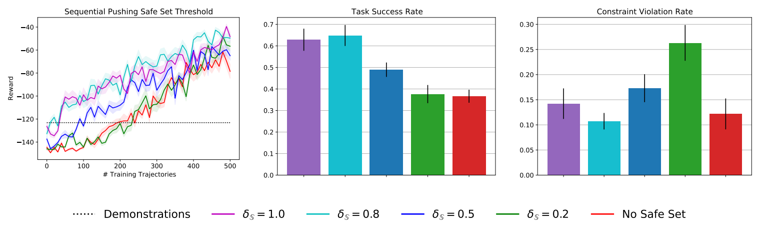


Figure F.3: **Hyperparameter Sweep for LS^3 Safe Set Threshold:** Plots show mean and standard error over 10 random seeds for experiments with different settings of $\delta_{\mathbb{S}}$ on the sequential pushing environment. We see that after offline training, the agent can successfully complete the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

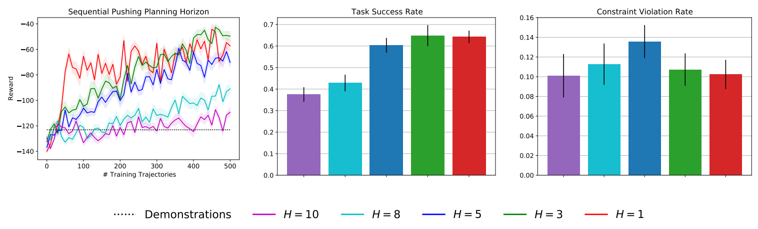


Figure F.4: **Hyperparameter Sweep for LS^3 Planning Horizon:** Plots show mean and standard error over 10 random seeds for experiments with different settings of H on the sequential pushing environment. We see that when the planning horizon is too high the agent cannot reliably complete the task due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

Appendix G

Monte Carlo Augmented Actor-Critic for Sparse Reward Deep RL from Suboptimal Demonstrations

G.1 Implementation Details

For all algorithms, all Q functions and policies are approximated using deep neural networks with 2 hidden layers of size 256. They are all updated using the Adam optimizer from [355].

G.1.1 Behavioral Cloning

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

G.1.2 Twin Delayed Deep Deterministic Policy Gradients

We use the author implementation of TD3 from [62], modifying it to implement MCAC. In order to maintain the assumption about complete trajectories described in Section 8.2, we modify the algorithm to only add to the replay buffer at the end of sampled trajectories, but continue to update after each timestep. We found the default hyperparameters from the repository to be sufficient in all environments.

G.1.3 Soft Actor Critic

For all SAC experiments we used a modified version of the SAC implementation from Tandon [350] which implements SAC with a double-Q critic update function to combat Q overestimation. Additionally, we modify the algorithm to satisfy the trajectory assumption as in

Section G.1.2. We mostly use the default hyperparameters from [193], but tuned α and τ. Parameter choices are shown in Table G.1.

Table G.1: Hyperparameters for SAC

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

G.1.4 Overcoming Exploration with Demonstrations

We implement the algorithm from [208] on top of the implementation of TD3 described in Section G.1.2. Because it would provide an unfair advantage over comparisons, the agent is not given the ability to reset to arbitrary states in the replay buffer. Since our setting is not goal-conditioned, our implementation does not include hindsight experience replay. For the value λ balancing the actor critic policy update loss and behavioral cloning loss, we use $\lambda = 1.0$.

G.1.5 Advantage Weighted Actor Critic

For AWAC experiments we use the implementation from [356], once again modifying it to maintain the assumption about complete trajectories and to implement MCAC. We found the default hyperparameter values to be sufficient in all settings.

Appendix H

Recovery RL: Safe Reinforcement Learning with Learned Recovery Zones

The supplementary material is structured as follows: In Section H.1 we discuss brief theoretical motivation for Recovery RL and possible variants and in Section H.2 we discuss algorithmic details for Recovery RL and comparison algorithms. In Section H.3, we report additional metrics for all domains and comparisons and in Section H.4 we visualize the safety critic for all navigation experiments. We provide additional details about algorithm implementation in Section H.5, and on domain-specific algorithm hyperparameters in Section H.6. Finally, we report simulation and physical environment details in Section H.7.

H.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.

H.1.1 Theoretical Motivation

Recall that the task policy is operating in an environment with modified dynamics:

$$P^{\pi_{\text{rec}}}_{\epsilon_{\text{risk}}}(s'|s,a) = \begin{cases} P(s'|s,a) & (s,a) \in \mathcal{T}^{\pi}_{\text{safe}} \\ P(s'|s,a^{\pi_{\text{rec}}}) & (s,a) \in \mathcal{T}^{\pi}_{\text{rec}} \end{cases} \tag{H.1}$$

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 $\mathcal{M}' = (\mathcal{S}, \mathcal{A}, P_{\epsilon_{\text{risk}}}^{\pi_{\text{rec}}}, R(\cdot,\cdot), \gamma, \mu)$, 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}.

H.1.2 Safety Value Function

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

H.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 $\mathcal{T}_{\text{safe}}^{\pi}$. While Q_{risk}^{π} in principle carries information about future safety, this is an alternative method to check future states.

H.2 Algorithm Details

H.2.1 Recovery RL

Recovery Policy: In principle, any off-policy reinforcement learning algorithm can be used to learn the recovery policy π_{rec}. 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 $\hat{Q}_{\phi,\text{risk}}^{\pi}(s, \pi_{\text{rec}}(s))$, as in the popular off-policy reinforcement learning algorithm DDPG [104]. 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:

$$L_\theta(s_t, a_t) = \mathbb{E}\left[\sum_{i=0}^{H} \hat{Q}_{\phi,\text{risk}}^{\pi}(\hat{s}_{t+i}, a_{t+i})\right] \tag{H.2}$$

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. [86] 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 $\hat{Q}_{\phi,\text{risk}}^{\pi}$ 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 [103] 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.

H.2.2 Unconstrained

We use an implementation of the popular model-free reinforcement learning algorithm Soft Actor Critic [350, 103], which maximizes a combination of task reward and policy entropy with a stochastic actor function.

H.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.

H.2.4 Risk Sensitive Policy Optimization (RSPO)

We implement Risk Sensitive Policy Optimization by implementing the Lagrangian Relaxation method as discussed in Section H.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 [235].

H.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 $Q^\pi_{\text{risk}}(s_t, a_t) \leq \epsilon_{\text{risk}}$.

H.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.

H.2.7 Off Policy Reward Constrained Policy Optimization (RCPO)

In on-policy RCPO [195], the policy is optimized via policy gradient estimators by maximizing $\mathbb{E}_\pi\left[\sum_{t=0}^\infty (\gamma^t R(s,a) - \lambda\gamma^t_{\text{risk}} D(s,a))\right]$. In this work, 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.

H.3 Additional Experimental Metrics

In Figure H.1 and Figure H.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 H.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 H.3 and H.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,

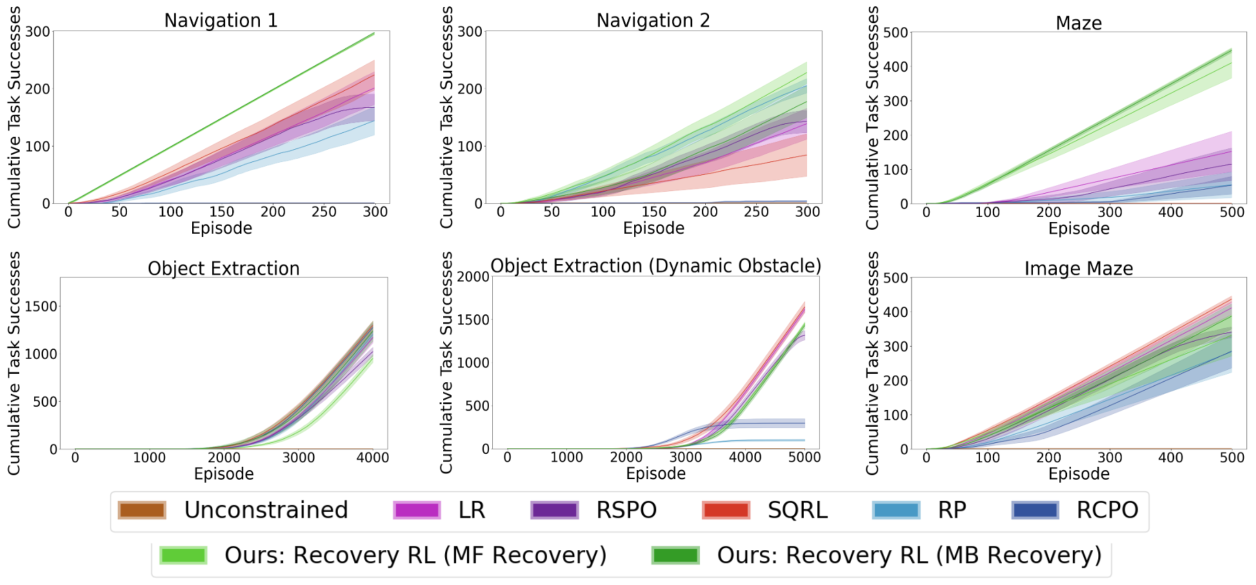


Figure H.1: **Simulation Experiments Cumulative Successes:** We plot the cumulative task successes for each algorithm in each simulation domain, with results averaged over 10 runs for all algorithms. We observe that Recovery RL (green), is generally among the most successful algorithms. In the cases that it has lower successes, we observe that it is safer (Figure H.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.

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 H.2, the results in Figure H.3 and Figure H.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 H.1, we report empirical results for when constraint violations occur in Table H.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 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.

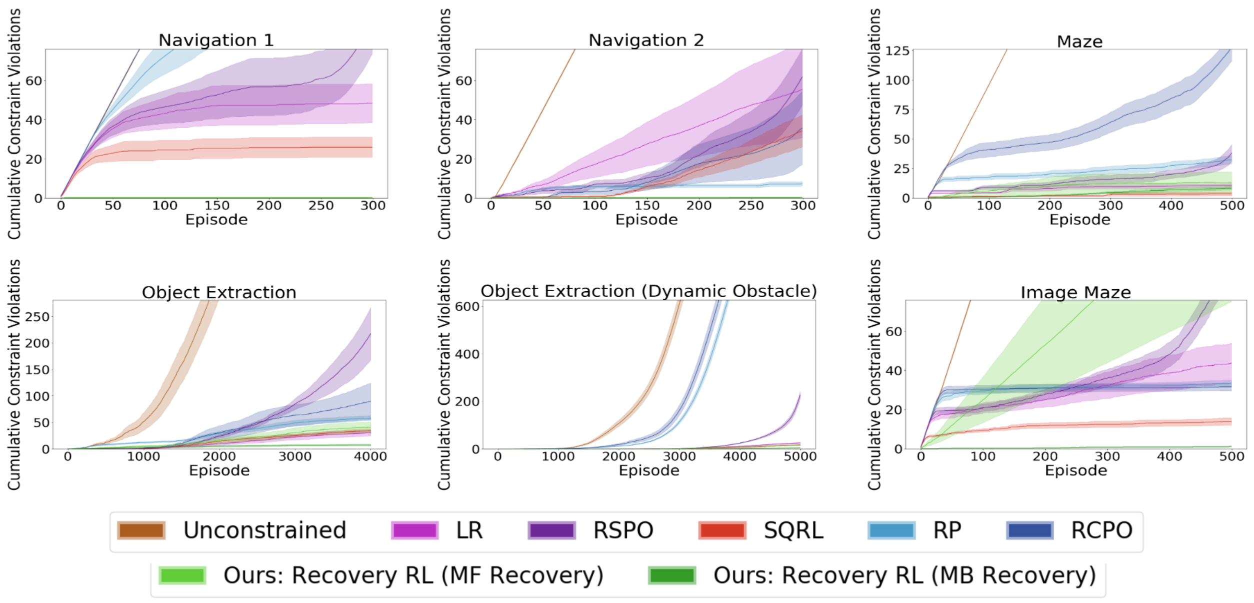


Figure H.2: **Simulation Experiments Cumulative Violations:** We plot the cumulative constraint violations for each algorithm in each simulation domain, with results averaged over 10 runs for all algorithms. We observe that Recovery RL (green), is among the safest algorithms across all domains. In the cases where it is less safe than a comparison, it has a higher task success rate (Figure H.1).

H.4 Safety Critic Visualizations

We visualize the safety critic after pretraining for the navigation domains in Figure H.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.

H.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, 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.

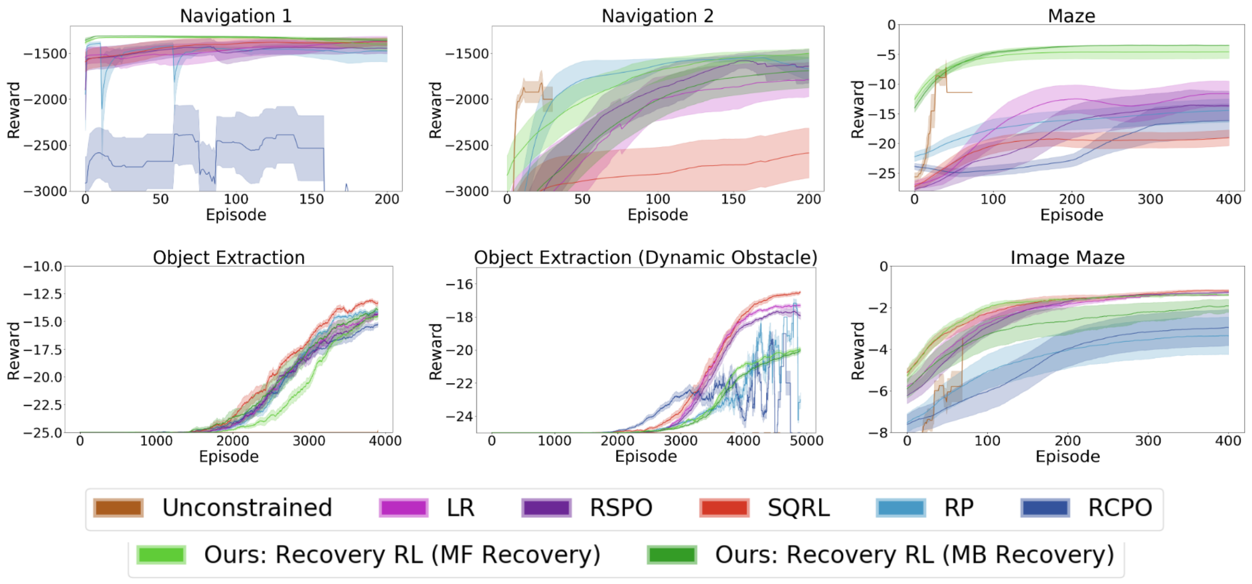


Figure H.3: **Simulation Experiments Reward Learning Curve:** We show the total reward attained in each episode smoothed over a 100 episode length window for each simulation domain, with results averaged over 10 runs for all algorithms. We do not show results when constraints are violated to prevent 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.

For all experiments, we build on the PyTorch implementation of Soft Actor Critic [228] provided in [350] 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 H.5.2 and then discuss the implementation of the recovery policy for Recovery RL in Section H.5.3.

H.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

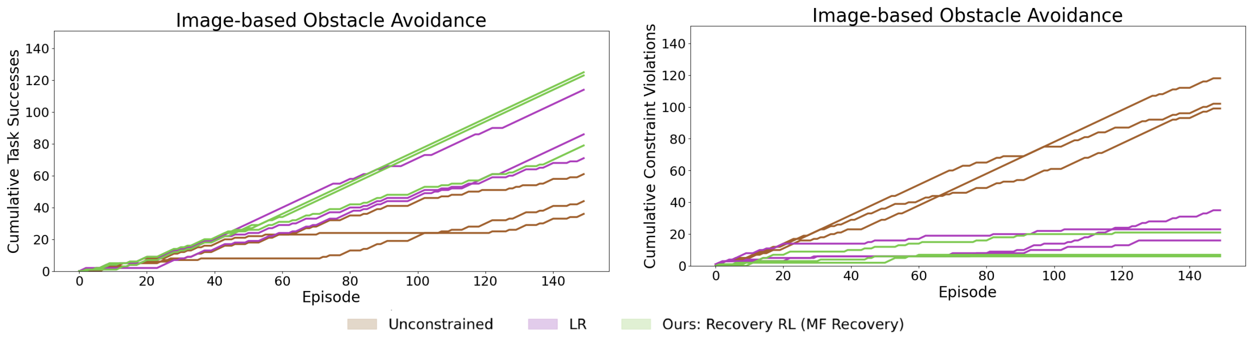


Figure H.4: **Physical Experiment Successes and Violations:** We plot the cumulative constraint violations and task successes for the image-based obstacle avoidance task on the dVRK for all 3 runs of each algorithm. We observe that Recovery RL is generally both more successful and safer than LR and unconstrained.

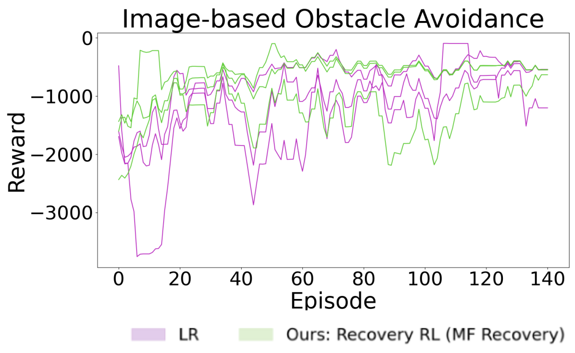


Figure H.5: **Physical Experiment Reward Learning Curve:** We show the total reward attained in each episode smoothed over a 10 episode length window with results from 3 runs for all algorithms. We do not show results when constraints are violated to prevent negative bias for algorithms which may violate 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.

Table H.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

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

H.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 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.

H.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 H.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 [103]. 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 [86, 357], 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. [86] 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, Savarese, and Finn [167]. 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, Savarese, and Finn [167]. For the encoder we utilize the following convolutional layers (channels, kernel size, stride): [(32, 4, 2), (32, 3, 1), (64, 4, 2), (64, 3, 1), (128, 4, 2), (128, 3, 1), (256, 4, 2), (256, 3, 1)] followed by fully connected layers of size $[1024, 512, 2L]$ where L is the size of the latent space (predict mean and variance). All layers use ReLU activations except for the last layer. The decoder takes a sample from the latent space of dimension L and then feeds this through fully connected layers $[128, 128, 128]$ which is followed by de-convolutional layers (channels, kernel size, stride):

Table H.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 H.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	$(\gamma_{\text{risk}}, \epsilon_{\text{risk}}, H)$

$[(128, 5, 2), (64, 5, 2), (32, 6, 2), (3, 6, 2)]$. 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.

H.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 H.3 and H.2.

H.7 Environment Details

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

H.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 [167]. 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.

H.7.2 Manipulation Environments

We build two manipulation environments on top of the cartgripper environment in the visual foresight repository [2]. 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.

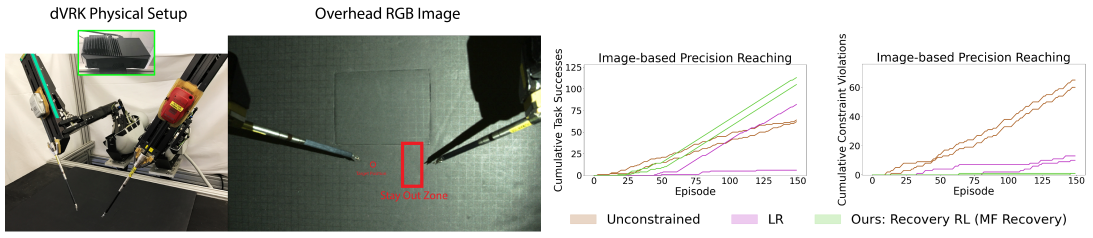


Figure H.6: **Additional Physical Experiment:** The image reacher task on the dVRK involves guiding the end effector to a target position while avoiding an invisible stay out zone in the center of the workspace. We plot the cumulative constraint violations and task successes the image reacher task on the dVRK. We observe that Recovery RL is both more successful and safer than LR and unconstrained.

H.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.

H.7.4 Physical Experiments

Physical experiments are run on the da Vinci Research Kit (dVRK) [105], 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 $48 \times 64 \times 3$. 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

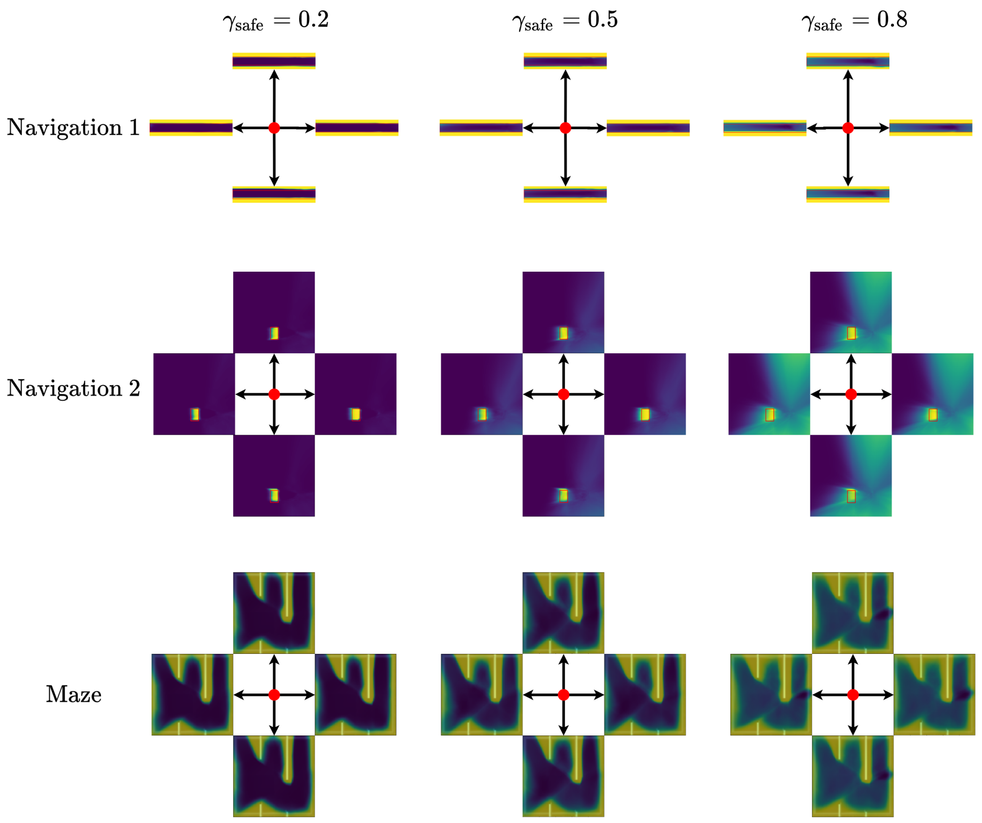


Figure H.7: $\hat{Q}^{\pi}_{\phi,\text{risk}}$ **Visualization:** We plot the safety critic Q^{π}_{risk} for the navigation environments using the cardinal directions (left, right, up, down) as action input. We see that as γ_{risk} is increased, the gradient is lower, and the the function more gradually 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.

effector position can be slightly inaccurate due to cabling effects such as hysteresis [248], but we ignore these effects in this work. 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.

H.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 H.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.

Appendix I

MESA: Offline Meta-RL for Safe Adaptation and Fault Tolerance

I.1 Algorithm Description

The full detail of the MESA algorithm is described in Algorithm 8. 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 [46] to avoid constraint violations.

I.2 Hyperparameters for MESA and Comparisons

We report global hyperparameters shared across all algorithms in Table I.1 and additionally include domain specific hyperparameters in separate tables in Tables I.2, I.3, and I.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 [103] and build on the implementation provided in [350].

I.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 $\sim$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

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 I.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 I.2: Navigation Hyperparameter Differences

Hyperparameters	Unconstrained	RRL	Multi-Task	MESA
Phase 1: Offline Training ($\mathcal{D}_{\text{train}}$)				
Total Iterations	—	—	15000	15000

Table I.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 I.4: Ant-Disabled Hyperparameter Differences.

datasets are generated, with each containing 200 episodes of data ($\sim$20K timesteps with 4.5K violations). The dataset from the testing environment contains 1K transitions (with 200 violations), which is **20x** smaller than before.

Dataset	N_{train}	$\lvert D_{\text{train}}\rvert$	$\lvert D_{\text{test}}\rvert$
Cartpole-Length	20	20K	1K
HalfCheetah-Disabled	4	400K	40K
Ant-Disabled	3	400K	40K

Table I.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.

Algorithm 8 MEta-learning for Safe Adaptation (MESA)

```
Require: Training datasets $\mathcal{D}^{train} = \{\mathcal{D}_i^{train}\}_{i=1}^{N_{\text{train}}}$, 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 $\beta$.
    {Phase 1: Offline Meta-Learning}
 1: for $i \in \{1, \dots N\}$ do
 2:   for $j \in \{1, \dots K\}$ do
 3:     Sample $\mathcal{D}_j^{train} \sim \mathcal{D}^{\text{train}}$
 4:     $\psi'_j \leftarrow \psi - \alpha_1 \cdot \nabla_\psi \mathcal{L}_{\text{risk}}\left(\psi, \mathcal{D}_j^{train}\right)$
 5:   end for
 6:   for $j \in \{1, \dots K\}$ do
 7:     Sample $\mathcal{D}_j^{test} \sim \mathcal{D}^{\text{test}}$
 8:   end for
 9:   $\psi \leftarrow \psi - \alpha_2 \cdot \sum_j \nabla_\psi \mathcal{L}_{\text{risk}}\left(\psi'_j, \mathcal{D}_j^{\text{train}}\right)$
10:   $\omega \leftarrow \omega - \beta \cdot \nabla_\omega \mathcal{L}_{\pi_{\text{rec}}}\left(\omega, \mathcal{D}^{\text{test}}\right)$
11: end for{Phase 2: Test Time Adaptation}
12: for $i \in \{1, \dots M\}$ do
13:   $\psi \leftarrow \psi - \alpha_1 \cdot \nabla_\psi \mathcal{L}_{\text{risk}}\left(\psi, \mathcal{D}^{\text{test}}\right)$
14:   $\omega \leftarrow \omega - \beta \cdot \nabla_\omega \mathcal{L}_{\pi_{\text{rec}}}\left(\omega, \mathcal{D}^{\text{test}}\right)$
15: end for
16: $\mathcal{D}^{\text{task}} \leftarrow \emptyset$ {Phase 3: Recovery RL}
17: while not converged do
18:   $s_1 \sim$ env.reset()
19:   for $t \in \{1, \dots H\}$ do
20:     $a_t^\pi, a_t^{\text{rec}} \sim \pi_\theta(\cdot|s_t), \pi_{\text{rec}}(\cdot|s_t)$
21:     if $Q_{\text{risk}}^\pi(s_t, a_t) \le \epsilon_{\text{risk}}$ then
22:       $a_t = a_t^\pi$
23:     else
24:       $a_t = a_t^{\text{rec}}$
25:     end if
26:     Execute $a_t$, observe $r_t$, $c_t$, and $s_{t+1}$
27:     Add $(s_t, a_t^\pi, c_t, s_{t+1})$ to $\mathcal{D}^{\text{test}}$
28:     Add $(s_t, a_t, r_t, s_{t+1})$ to $\mathcal{D}^{\text{task}}$
29:     $\theta \leftarrow \theta - \gamma \cdot \nabla_\theta \mathcal{L}_\pi\left(\theta, \mathcal{D}^{\text{task}}\right)$
30:     $\psi \leftarrow \psi - \alpha_1 \cdot \nabla_\psi \mathcal{L}_{\text{safe}}\left(\psi, \mathcal{D}^{\text{test}}\right)$
31:     $\omega \leftarrow \omega - \beta \cdot \nabla_\omega \mathcal{L}_{\pi_{rec}}\left(\omega, \mathcal{D}^{\text{test}}\right)$
32:     if $c_t$ then
33:       End episode
34:     end if
35:   end for
36: end while
```

Appendix J

Exploratory Grasping: Asymptotically Optimal Algorithms for Grasping Challenging Polyhedral Objects

The supplementary material is organized as follows. In Section J.1 we provide proofs for all theoretical results in Sections 12.3 and 12.4. In Section J.2 we provide additional experimental details, in Section J.3 we include additional simulation experiments, and in Section J.4 we perform sensitivity experiments. In Section 12.5.2 we provide further details on physical experiments.

J.1 Proofs

Here we provide the proofs for all results in Section 12.3 and Section 12.4 in the main text.

J.1.1 Proof of Lemma 12.3.1

Consider Exploratory Grasping MDP $\mathcal{M}'$ for which the object stable pose does not change when a grasp fails. Thus, it must be the case that $\delta_{l,m} = 0$ when $l \neq m$ and $\delta_{l,l} = 1\ \forall l$. Note that for any Exploratory Grasping MDP $\mathcal{M}$, it must be the case that $D(\mathcal{M}) \leq D(\mathcal{M}')$ since the probability of transition between any pair of distinct states in $\mathcal{M}'$ is at most the probability of transition in $\mathcal{M}$. Now we establish a bound on $D(\mathcal{M})$ by bounding $D(\mathcal{M}')$.

Without loss of generality, let $\lambda_1 \leq \lambda_s\ \forall s \in \mathcal{S}$. Additionally, define π^* as the policy which selects the grasp with highest success probability on all poses, with associated probability transition matrix P^{π^*} and with hitting time $T^{\pi^*}_{s \to s'}$ defined as in Definition 12.3.1. Then, the diameter of $\mathcal{M}'$ can be computed as follows.

For MDP $\mathcal{M}'$, it must be the case that

$$\min_{\pi} T^{\pi}_{s\to s'} = T^{\pi^*}_{s\to s'} \tag{J.1}$$

since π^*, the policy which always picks the highest quality grasp on each pose, minimizes the hitting time between any pair of poses s, s'. Furthermore, note that

$$\max_{s\neq s'} T^{\pi^*}_{s\to s'} = \max_{s} T^{\pi^*}_{s\to 1} \tag{J.2}$$

since for any starting pose s, the hitting time between s and s' will always be highest for $s' = 1$ (the pose with lowest drop probability) for any policy π. Thus, we see that

$$D(\mathcal{M}') = \max_{s} T^{\pi^*}_{s\to 1} \tag{J.3}$$

Finally, we leverage equation J.3 to compute an upper bound on $D(\mathcal{M}')$ as follows.

Let π_ϵ be any policy which selects a grasp with success probability ϵ on each stable pose l. Note that by Assumption 12.2.3 we assume that there exists a grasp with success probability at least ϵ on each pose. Without loss of generality, we can consider the case that there exists a grasp with success probability exactly ϵ on each pose and the policy which selects these grasps since the hitting times under this policy will only be greater than those under a policy which selects grasps with success probability greater than ϵ. Then, it follows that

$$\min_{\pi} T^{\pi}_{s\to s'} \leq T^{\pi_\epsilon}_{s\to s'} \ \forall s, s' \tag{J.4}$$

Now note that since we are considering pose evolution under π_ϵ, which selects a grasp of the same quality ϵ on any pose, the starting pose s does not affect the hitting time. Combining this with the fact that the hitting time to the least likely pose (pose 1) will always be the highest for any policy π and inequality (J.4) yields that

$$D(\mathcal{M}') \leq \max_{s\neq s'} T^{\pi_\epsilon}_{s\to s'} = T^{\pi_\epsilon}_{2\to 1} \tag{J.5}$$

Since the choice of s does not matter under π_ϵ, we use $s = 2$ above without loss of generality. Now, note that the hitting time to pose 1 under π_ϵ is distributed as a geometric random variable with parameter $\epsilon\lambda_1$, which has mean $\frac{1}{\epsilon\lambda_1}$, yielding the desired result. $\square$

J.1.2 Proof of Theorem 12.3.1

The result immediately follows from combining the diameter bound from Lemma 12.3.1 and the regret bounds established for UCRL2 [333], KL-UCRL [335] and PSRL [338] (average regret $\tilde{O}(DS\sqrt{A/T})$) and for UCRLV [334] (average regret $\tilde{O}(\sqrt{DSA/T})$) where D is the MDP diameter and S and A are the cardinalities of the state space and action space respectively. $\square$

J.1.3 Proof of Theorem 12.4.1

We bound the expected regret of BORGES by decomposing the regret into two terms, one which depends on the divergence between the distribution of poses seen by the optimal policy and $\pi^{\mathcal{B}}$, and the other of which depends on the difference in rewards attained by $\pi^{\mathcal{B}}$ and π^* when evaluated on the distribution of poses seen by $\pi^{\mathcal{B}}$. For simplicity, we refer to $\pi^{\mathcal{B}}$ as π for the proof.

$$\begin{aligned}
\mathbb{E}\left[\mathcal{R}^{\mathcal{B}}(T)\right] &= \sum_{s=1}^{S} p_T^*(s)\mathbb{E}\left[\frac{1}{T_s^*}\sum_{t=1}^{T_s^*} R(s,\pi^*(s))\right] - \sum_{s=1}^{S} p_T^{\pi}(s)\mathbb{E}\left[\frac{1}{T_s^{\pi}}\sum_{t=1}^{T_s^{\pi}} R(s,\pi_t(s))\right] && \text{(J.6)}\\
&= \sum_{s=1}^{S}\left(p_T^*(s)-p_T^{\pi}(s)\right)g_s^*(T_s^*) + \sum_{s=1}^{S} p_T^{\pi}(s)\left(g_s^*(T_s^*)-g_s^{\pi}(T_s^{\pi})\right) && \text{(J.7)}\\
&= \mathbb{E}\left[\mathcal{R}_{\pi}^{\mathcal{B}}(T)\right] + \sum_{s=1}^{S}\left(p_T^*(s)-p_T^{\pi}(s)\right)g_s^*(T_s^{\pi}) && \text{(J.8)}
\end{aligned}$$

where (J.7) follows from letting $g_s^*(T_s^*) = \mathbb{E}\left[\frac{1}{T_s^*}\sum_{t=1}^{T_s^*} R(s,\pi^*(s))\right]$ and $g_s^{\pi}(T_s^{\pi}) = \mathbb{E}\left[\frac{1}{T_s^{\pi}}\sum_{t=1}^{T_s^{\pi}} R(s,\pi_t(s))\right]$ and (J.8) follows from letting $\mathbb{E}\left[\mathcal{R}_{\pi}^{\mathcal{B}}(T)\right]$ denote the expected regret on the distribution of poses visited by policy π and noting that the average reward for the optimal policy, g_s^*, is independent of the timesteps spent in the pose (i.e., $g_s^*(T_s^{\pi}) = g_s^*(T_s^*)\ \forall s$).

We first focus on the first term in (J.8). We know that $\mathbb{E}\left[\mathcal{R}_{\pi}^{\mathcal{B}}(T)\right]$ approaches zero if each pose is visited infinitely often in the limit as $T \to \infty$ provided that $\mathcal{B}$ is a no-regret online learning algorithm:

$$\lim_{T\to\infty} p_T^{\pi}(s) > 0, \forall s \in \{1,2,\ldots,S\} \quad \Rightarrow \quad \lim_{T\to\infty}\mathbb{E}\left[\mathcal{R}_{\pi}^{\mathcal{B}}(T)\right] = 0 \tag{J.9}$$

Thus, it remains to be shown that under π, all poses are visited infinitely often in the limit as $T \to \infty$. Note that this statement is equivalent to showing that in the limit as $T \to \infty$, bandit algorithm $\mathcal{B}$ selects grasps on each pose with non-zero success probability with non-zero probability. Suppose that this was not the case (i.e., that as $T \to \infty$, $\mathcal{B}$ assigns zero grasp probability to all grasps with non-zero success probability). This would imply that $\mathcal{B}$ only selects grasps with zero success probability, and thus incurs constant regret on its own distribution ($\lim_{T\to\infty}\mathbb{E}\left[\mathcal{R}_{\pi}^{\mathcal{B}}(T)\right] > 0$). This contradicts the initial assumption that $\mathcal{B}$ is a no-regret online learning algorithm, showing that under π, all poses must be visited infinitely often in the limit as $T \to \infty$.

Now we shift our attention to the second term in (J.8). Given that $\mathcal{B}$ is a no-regret online learning algorithm, it must be the case that $g_s^{\pi}(T_s^{\pi}) \xrightarrow[T_s^{\pi}\to\infty]{} g_s^*(T_s^{\pi})\ \forall s$. This implies that in the limit as $T \to \infty$, π and π^* have the same success rate on all stable poses. Two policies with the same success rate on all stable poses induce the same Markov chain over $\mathcal{S}$, and thus admit the same stable pose distribution. Thus, $p_T^{\pi}(s) \xrightarrow[T\to\infty]{} p_T^*(s)$, implying that the second term also approaches 0 as $T \to \infty$. □

J.1.4 Proof of Theorem 12.4.2

Let X denote the random variable that is the number of drops needed until we've reached every stable pose. We assume we have n distinct stable poses. Let X_k be the discrete random variable that represents the number of drops after visiting the $(k-1)$th distinct stable pose to visit the kth distinct stable pose. As a base case, $X_1 = 1$ since the first pose we drop into will always be distinct. By linearity of expectation we have

$$\mathbb{E}[X] = \sum_{j=1}^{N} \mathbb{E}[X_j]. \tag{J.10}$$

After we visit the $(j-1)$th distinct stable pose we have $N-j+1$ stable poses left that haven't been visited yet. As before, we order the stable poses in order from most likely to least likely: $\lambda_1 > \lambda_2 > \cdots > \lambda_N$. Because the expected number of drops to get into an unseen pose is a geometric random variable, we can upper bound the number of drops needed to visit all stable poses as

$$\mathbb{E}[X] = \sum_{j=1}^{N} \mathbb{E}[X_j] \leq \sum_{j=1}^{N} \frac{1}{1-\sum_{i=2}^{j} \lambda_{i-1}}. \tag{J.11}$$

We derive this by first noting that the probability of visiting a new stable pose is $1 - \sum_{i \in \text{visited}} \lambda_i$ and the expected number of drops to successfully visit a new stable pose is a geometric random variable. We want to bound the number of drops to visit all stable poses, but we do not know what order we will visit these poses. However, the worst-case outcome (largest number of required drops) will be if we visit them in order from largest drop probability to smallest since this will reduce the probability as quickly as possible which will maximize $E[X]$ since the probabilities are in the denominator. The above analysis assumes we are repeatedly able to randomly drop the object, with no accounting for grasp success or failure. Rather than only bounding the number of drops, we want to bound the expected number of grasps to visit all stable poses at least once.

The grasping strategy will impact how often we need to attempt grasps before we get to drop again. For a worst-case analysis, we assume that there are no topples, $\delta_{s,s'} = 0 \ \forall s, s'$, and assume that each stable pose has only one grasp with non-zero success probability. We will also let ϵ be the lower bound on the success probability of this grasp across all stable poses. If we just pick grasps uniformly at random on each pose, we have a $1/K$ probability of selecting the best grasp (ie. the one grasp with non-zero success probability) out of K possible grasps. Note that this strategy can only yield strictly lower grasp success rate than BORGES since any no-regret algorithm (such as UCB or Thompson sampling) will cause BORGES to pick grasps with non-zero success probability with higher probability than grasps which never succeed. Thus, if we can upper bound the number of grasps under a uniform random grasp strategy this will upper bound the number of grasps for BORGES. The lower bound on the probability of a successful grasp which leads to a redrop is ϵ. Thus, in the worst-case, we have probability ϵ/K of getting to redrop and the worst-case expected number of grasps to get a successful grasp is K/ϵ since this is a geometric random variable. Thus, the expected number of grasps we need to visit all stable poses is no more than

$$\frac{K}{\epsilon}\sum_{j=1}^{N}\frac{1}{1-\sum_{i=2}^{j}\lambda_{i-1}}. \tag{J.12}$$

□

J.2 Experimental Details

Object Selection: We choose 7 objects from the set of adversarial objects in Dex-Net 2.0 [300] because these objects had empirically been shown to be difficult to grasp for the Dex-Net policy. Similarly, the recently-introduced EGAD! object dataset [341] was created to contain objects with few high-quality parallel-jaw grasps. For this dataset, we select all objects for which there exists at least one sampled grasp of quality $\epsilon = 0.1$ on at least one stable pose of the object. Of the 49 objects in the EGAD evaluation dataset, 39 met this criterion.

Pose Selection: For each of the objects, we remove stable poses from the distribution in simulation if they occur with less than a 0.1% chance or if they do not contain a sampled grasp with quality at least $\epsilon = 0.1$. When a pose is removed, the remaining stable pose distribution is renormalized.

Grasp Sampling: We sample a set of $K = 100$ parallel-jaw grasps on the image observation of each pose of each object as in [300]. This sampling process is done using the depth image grasp sampler from the GQCNN repository and is repeated for up to 10 iterations. At each iteration, the sampled grasps' ground truth qualities are calculated using the robust wrench resistance metric that measures the ability of the grasp to resist gravity [308]. If no grasps are found with quality of at least $\epsilon = 0.1$, then the sampling process is repeated for another iteration where more grasps are sampled. If a grasp is found with quality at least ϵ, then that grasp is selected along with 99 other grasps chosen at random from the sampled grasps. If the maximum number of iterations are exceeded without finding a grasp with quality ϵ, the stable pose is discarded.

J.3 Additional Simulation Experiments

We also evaluate BORGES and baselines in a setting in which toppling is not possible ($\delta_{s,s'} = 0$, $\forall s \neq s$). This exacerbates the difficulty of grasp exploration since an object must be successfully grasped in a given pose for policies to be able to explore grasps in other poses. When evaluating policies, we remove poses that have no grasps with nonzero ground-truth quality and renormalize the stable pose distribution. We emphasize that we perform this step for all policies and do this only to ensure that no poses act as sink states from which the robot can never change the pose in the future (and thus ensure that Assumption 12.2.3 is satisfied). For the 7 objects from the Dex-Net 2.0 adversarial object dataset, 11% of poses (an average of 1.3 poses per object) were removed. These poses made up an average of 2.8% of the probability mass of the stable pose distribution. These results are shown in Figure J.1. We find that BORGES again significantly outperforms baseline policies in this setting.

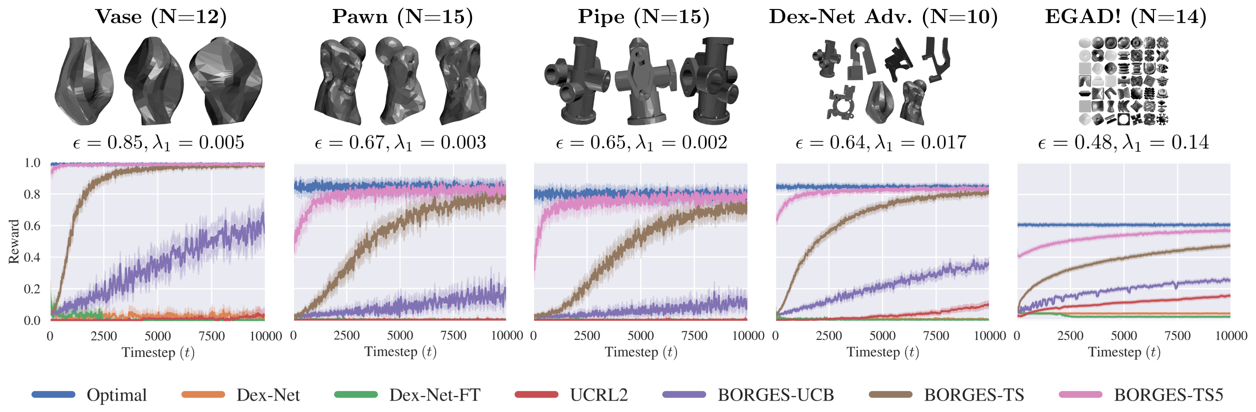


Figure J.1: Reward plots for three Dex-Net Adversarial objects as well as aggregated rewards across all Dex-Net Adversarial objects and EGAD! objects in a scenario where no toppling is allowed between poses. Poses without any nonzero quality grasps are discarded. When transitions can only occur due to successful grasps, Exploratory Grasping policies converge to the optimal policy much more quickly than baselines.

We show the importance of Assumption 12.2.3 even when toppling is included by evaluating policies on all stable poses of the object, regardless of whether a grasp with nonzero ground-truth quality exists or toppling out of each pose is possible. Thus, it is possible that some poses may act as sink states. Figure J.2 suggests that for objects that do not obey Assumption 12.2.3, such as the pawn, all policies find sink states and remain in them. For objects that do obey Assumption 12.2.3, but have stable poses that can only transition via toppling, such as the mount, performance decreases for all policies, but the order of convergence is consistent.

J.4 Sensitivity Experiments

J.4.1 Sensitivity to Number of Grasp Samples

We vary K, the number of grasp samples per object pose, within the range $K = \{10, 20, 50, 100, 200\}$ across the 7 objects in the Dex-Net Adversarial object set. The results in Figure J.3 suggest that as K increases, the likelihood of sampling a high-quality grasp on each pose also

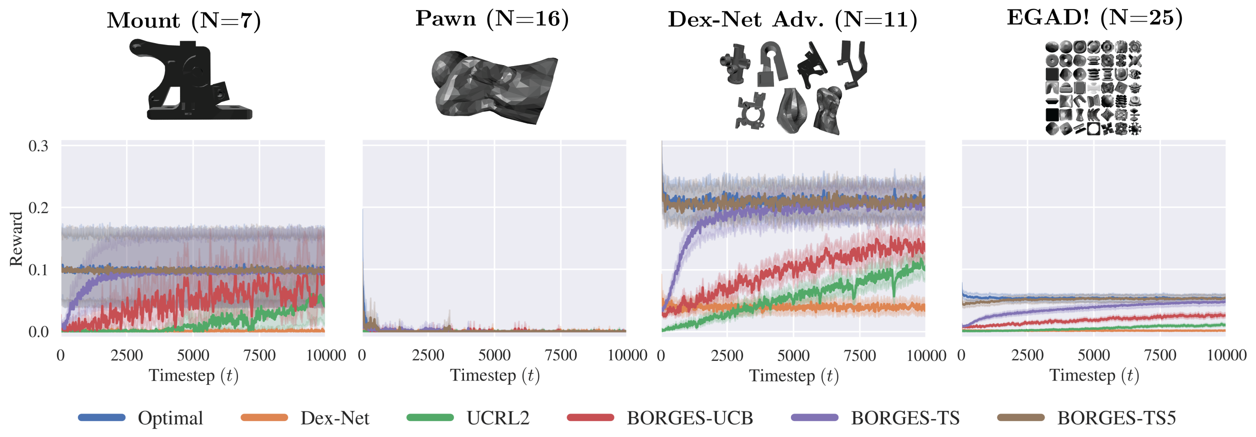


Figure J.2: Reward plots for two Dex-Net adversarial objects shown alongside an aggregation of all Dex-Net adversarial objects and EGAD! objects in a scenario where no poses are removed from exploration. In such a case, there is a split between objects without sink states, such as the mount, and objects with sink states, such as the pawn, where no means of transitioning via toppling or grasping is available.

increases. With more grasps to consider, each policy takes longer to converge to the optimal policy, but the order of convergence between policies is consistent.

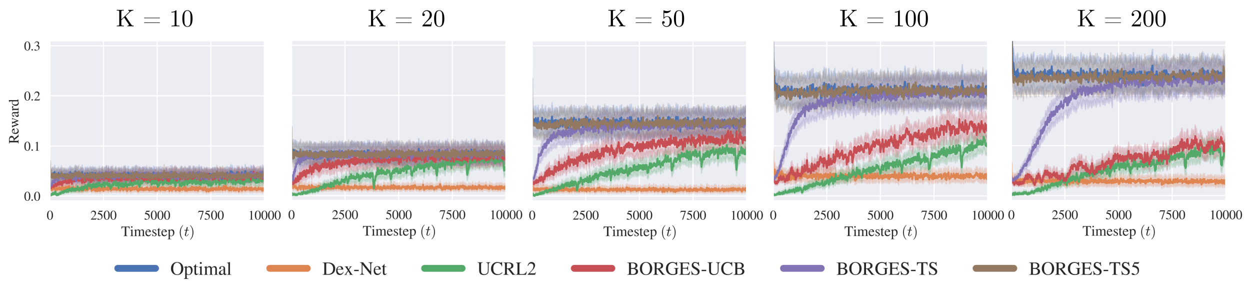


Figure J.3: Sensitivity of BORGES to the number of grasp samples K across the 7 Dex-Net Adversarial objects. With more grasp samples, the likelihood of including a higher-quality grasp increases, but the rate of convergence decreases.

J.4.2 Sensitivity to Exploratory Grasping Parameters

We perform sensitivity analysis of BORGES to the Exploratory Grasping parameters ϵ and λ_1. For these experiments, we evaluate the policy using a set of synthetic objects with $\lambda_1 = \{0.001, 0.01, 0.1, 0.2\}$ and $\epsilon = \{0.1, 0.25, 0.5, 0.75, 1.0\}$ and for simplicity consider the case in which toppling is not possible. In each case, we choose a single grasp on each pose to have reliability ϵ with all other grasps having a mean parameter of 0. The results are shown in Figure J.4. These results suggest that unless ϵ or λ_1 are low, BORGES quickly converges to the optimal policy. In particular, ϵ has an outsized effect on the accumulated reward; with $\epsilon \leq 0.10$, we observe that the policy fails to approach the optimal policy regardless of λ_1 through the first 10,000 timesteps.

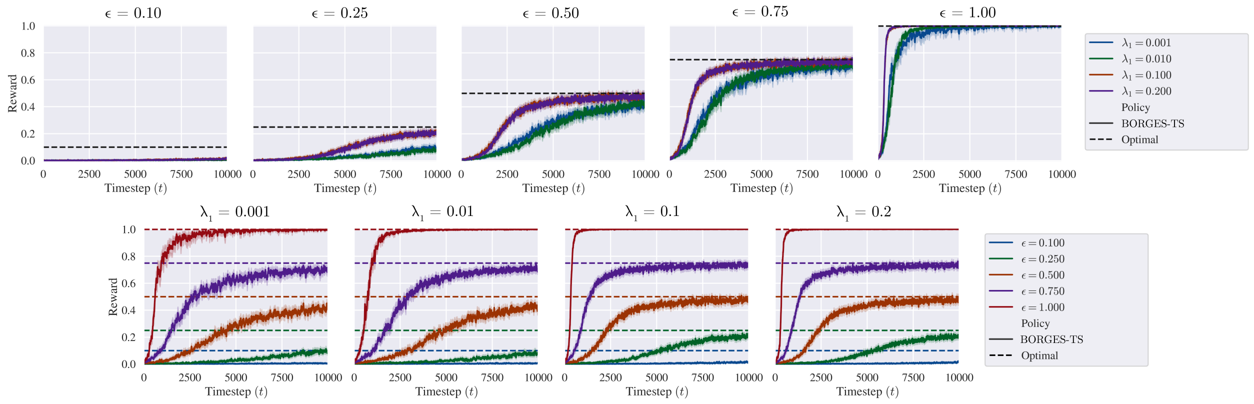


Figure J.4: Sensitivity of BORGES to the Exploratory Grasping parameters ϵ and λ_1, as shown across 20 synthetic objects. Unless ϵ or λ is low, BORGES quickly converges to the optimal policy. However, when either is low, particularly ϵ, the policy converges much more slowly, taking even more than the 10,000 timesteps shown for some combinations.

J.5 Initial Physical Experiments Details

In physical experiments, we evaluate BORGES on two challenging 3D printed objects and compare performance with the Dex-Net policy from Section 12.5.1. The object is placed in front of an ABB Yumi robot with a parallel-jaw gripper and overhead Photoneo PhoXi S camera. At each timestep, we capture a depth image observation from the camera and estimate if the current object stable pose has been seen previously with the following stable pose change detection procedure. Specifically, we cache the first depth image seen for each new pose, and compare the current depth image at each timestep to the cached depth images. We segment out the background and project the depth image into a point cloud representation. We then mean-center the current point cloud, apply 7200 rotations around the z-axis, and measure the chamfer distance between each resulting rotated point cloud and the point cloud representations of each of the cached depth images. We classify each point in the current point cloud as an inlier if the closest point in a previous point cloud is less than 0.02 mm away. If at least 80% of the points are inliers, we classify the two point clouds as belonging to the same stable pose. If the pose has not been seen previously, we add the depth image to the cache, sample a set of grasps, and calculate the Dex-Net predicted quality values for each pose. Otherwise, we recall the previously sampled grasps for the given pose

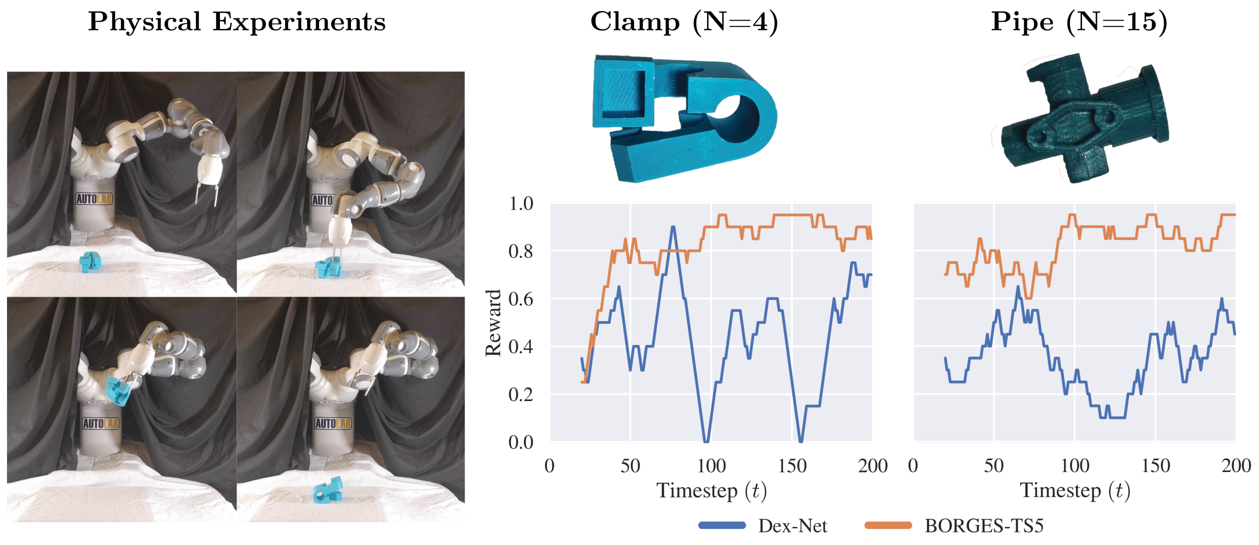


Figure J.5: Experiment setup and learning curves for the Dex-Net and BORGES-TS5 policies for the clamp and pipe objects across 200 grasp attempts (smoothed with a running average of 20 attempts). The robot attempts to grasp the object at each timestep and, if it succeeds, rotates and drops the object to sample from the stable pose distribution (left). BORGES-TS5 quickly converges within 100 attempts on both objects, indicating that it finds grasps that succeed nearly every time for each pose. Dex-Net's performance remains uneven, indicating that it finds high-quality grasps for some poses, but not others.

and transform them by the relative transform between the cached and current images. A grasp is executed according to the Dex-Net or BORGES policy; if the grasp is successful, the object is lifted, rotated by 30 degrees around an axis sampled uniformly at random from 3D unit vectors, and released. If the grasp is unsuccessful, the object may topple into another pose or may remain in the same pose. This process repeats for 200 timesteps per object. After each grasp attempt, we move the object to the center of the workspace to prevent errors in the stable pose change detection procedure, but strongly suspect that this procedure will likely be unnecessary with a more constrained workspace or a more robust stable pose change detection procedure, which we will implement in future work.

In Figure J.5 we show a larger version of Figure 12.2 from Section 12.5.2. As described in Section 12.5.2, we find that BORGES is able to significantly increase the grasp success rate of a Dex-Net policy, achieving success rates of 0.89 and 0.87 on the clamp and pipe, respectively, as compared to 0.49 and 0.37 for the Dex-Net policy with just 200 grasp attempts in the real world. This effect highlights the importance of a policy that is able to learn online from successful and failed grasp attempts; Dex-Net does not learn online so continuing to attempt grasps will lead to the same high-variance behavior over time, but BORGES continues to stabilize as it approaches optimal performance across all poses.