Accelerating Robot Learning and Deformable Manipulation Using Simulated Interactions, Architectural Priors, and Curricula

by

Daniel Seita

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor John Canny, Co-chair
Professor Ken Goldberg, Co-chair
Professor Masayoshi Tomizuka

Summer 2021

Accelerating Robot Learning and Deformable Manipulation
Using Simulated Interactions, Architectural Priors, and Curricula

Abstract

Accelerating Robot Learning and Deformable Manipulation
Using Simulated Interactions, Architectural Priors, and Curricula

by

Daniel Seita

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor John Canny, Co-chair

Professor Ken Goldberg, Co-chair

The robotics community has seen tremendous advances in grasping and manipulating a wide variety of objects using learning-based techniques. Deep learning approaches are often sample inefficient, and most work deals with grasping rigid objects rather than deformable objects such as ropes, clothing, and bags, which are ubiquitous in our daily lives. In this thesis, I will present novel approaches that can result in more efficient robot learning, including learning from simulation by using a skilled supervisor or building a dynamics model, learning using action-centric neural network architectures, and learning using a curriculum of samples. I will demonstrate results on manipulating deformable objects in simulation and real settings, and learning policies for Atari and MuJoCo environments.

To my family. *We did it.*

Contents

Contents **ii**

1 Introduction **1**
1.1 Deformable Object Manipulation 1
1.2 Deep Learning for Robotics 2
1.3 Contributions of this Thesis 3
1.4 Thesis Structure 4

I Case Study: Robot Bed-Making **6**

2 Deep Transfer Learning of Pick Points on Fabric for Robot Bed-Making **7**
2.1 Related Work 8
2.2 Methodology 10
2.2.1 Setup 10
2.2.2 Grasp Network for Pick Points 10
2.3 Data and Experiments 11
2.3.1 Data Collection and Processing 11
2.4 Results 13
2.4.1 Grasp Network Training 13
2.4.2 Harris Corner Detector is Insufficient 14
2.4.3 Physical Robot Deployment Evaluation 15
2.4.4 Timing Analysis 20
2.5 Conclusion and Future Work 20

II Manipulating Fabrics from Simulation **22**

3 Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor **23**
3.1 Related Work 23
3.1.1 Reinforcement Learning for Fabric Manipulation 24

3.1.2 Fabric Smoothing . . . 25
3.2 Problem Statement . . . 25
3.3 Fabric and Robot Simulator . . . 26
3.3.1 Actions . . . 28
3.3.2 Starting State Distributions . . . 28
3.4 Baseline Policies . . . 28
3.5 Simulation Results for Baseline Policies . . . 30
3.6 Imitation Learning with DAgger . . . 32
3.6.1 Policy Training Procedure . . . 33
3.6.2 Simulation Experiments . . . 33
3.7 Physical Experiments . . . 34
3.7.1 Physical Experiment Protocol . . . 35
3.7.2 Physical Experiment Results . . . 35
3.7.3 Failure Cases . . . 36
3.8 Conclusion and Future Work . . . 37

4 VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation 39
4.1 Related Work . . . 41
4.2 Problem Statement . . . 42
4.3 Approach . . . 42
4.3.1 Goal Conditioned Fabric Manipulation . . . 42
4.3.2 Fabric Simulator . . . 43
4.3.3 Data Generation . . . 45
4.3.4 VisuoSpatial Dynamics Model . . . 45
4.3.5 VisuoSpatial Foresight . . . 46
4.4 Simulated Experiments . . . 46
4.4.1 VisuoSpatial Dynamics Prediction Quality . . . 46
4.4.2 Fabric Smoothing Simulations . . . 48
4.4.3 Fabric Folding Simulations . . . 49
4.5 Physical Experiments . . . 51
4.5.1 Experiment Protocol . . . 51
4.5.2 Physical Fabric Smoothing . . . 52
4.5.3 Physical Fabric Folding . . . 55
4.6 Conclusion and Future Work . . . 56

III Benchmarks and Architectures 57

5 Learning to Rearrange Deformable Cables, Fabrics, and Bags with Goal-Conditioned Transporter Networks 58
5.1 Related Work . . . 59
5.1.1 Deformable Object Manipulation . . . 59

5.1.2 Data-Driven Robot Manipulation . . . 61
5.2 Background . . . 61
5.2.1 Problem Formulation . . . 61
5.2.2 Background: Transporter Networks . . . 62
5.3 Goal-Conditioned Transporter Networks . . . 64
5.3.1 Goal-Conditioned Transporter Networks . . . 64
5.3.2 Training Details for Goal Conditioned Models . . . 64
5.4 Simulator and Tasks . . . 65
5.4.1 Deformable Objects (Soft Bodies) in PyBullet . . . 66
5.4.2 Benchmark for Manipulating Deformable Objects . . . 66
5.5 Experiments . . . 67
5.6 Results . . . 68
5.6.1 Non-Goal Conditioned Tasks . . . 70
5.6.2 Goal Conditioned Tasks . . . 70
5.7 Conclusion and Future Work . . . 70

IV Curriculum Learning 72

6 ZPD Teaching Strategies for Deep Reinforcement Learning from Demonstrations 73
6.1 Related Work . . . 74
6.2 Methodology . . . 75
6.2.1 Background . . . 75
6.2.2 The Student and Teacher . . . 76
6.2.3 Student/Teacher Matching with f_{select} . . . 77
6.2.4 Minibatch Sampling with f_{blend} . . . 78
6.3 Experimental Setup . . . 78
6.4 Results . . . 80
6.4.1 Per-Game Analysis . . . 83
6.5 Conclusion and Future Work . . . 84

7 DCUR: Data Curriculum for Teaching via Samples with Reinforcement Learning 86
7.1 Related Work . . . 87
7.2 Problem Statement and Preliminaries . . . 89
7.3 Methodology . . . 89
7.3.1 Teachers and Data Generation . . . 89
7.3.2 Data CUrriculum for Reinforcement Learning (DCUR) Framework . . . 91
7.3.3 Apprenticeship Learning: Using a Small Amount of Online Data . . . 92
7.4 Experiments . . . 92
7.5 Results . . . 94

7.5.1 Effect of Data Curricula on Student Training Offline 94
7.5.2 Data Curriculum for Training 10X Longer 95
7.5.3 Apprenticeship Learning: Results with Small Amounts of Online Data 96
7.5.4 Investigation of Q-Values 97
7.6 Conclusion and Future Work 98

V Conclusion and Future Work 99

8 Conclusion 100
8.1 Future Research Opportunities 101
8.2 The Long-Term Vision 104

Bibliography 105

A Appendix for Chapter 2 122
A.1 Additional Experiment Information 122
A.2 Additional Qualitative Results 123

B Appendix for Chapter 3 125
B.1 Fabric Environment 125
B.1.1 Actions 125
B.1.2 Starting State Distributions 127
B.2 Details on Baseline Policies 127
B.2.1 Highest (Max z) 128
B.2.2 Wrinkles 128
B.2.3 Oracle 128
B.2.4 Oracle Expose 128
B.3 Details on Imitation Learning 129
B.3.1 DAgger Pipeline 129
B.3.2 Domain Randomization and Simulated Images 130
B.4 Experiment Setup Details 132
B.4.1 Image Processing Pipeline 132
B.4.2 Physical Experiment Setup and Procedures 133
B.5 Additional Experiments 133

C Appendix for Chapter 4 136
C.1 Fabric Simulators 136
C.2 Details of Learning-Based Methods 137
C.2.1 Imitation Learning Baseline: DAgger 137
C.2.2 Model-Free Reinforcement Learning Baseline: DDPG 139
C.2.3 VisuoSpatial Foresight 142
C.3 Supplementary Smoothing Results 143

C.3.1 Statistical Significance Tests . . . 143
C.3.2 Domain Randomization Ablation . . . 143
C.3.3 Action Magnitudes . . . 144

D Appendix for Chapter 5 **146**
D.1 COVID-19 Details . . . 146
D.2 Task Details . . . 147
D.2.1 Preliminaries . . . 147
D.2.2 DeformableRavens Task Details . . . 148
D.2.3 Additional Task: Block-Notarget . . . 149
D.3 Demonstrator Data Policy . . . 150
D.4 Additional Transporter Network Details . . . 152
D.5 Additional Experiment Details . . . 153
D.6 Additional Experiment Results . . . 154

E Appendix for Chapter 7 **160**
E.1 Environments and Teacher Performance . . . 160
E.2 Additional Experiment Details . . . 161
E.2.1 Evaluation and Judging Performance with Standard Errors . . . 162
E.3 Additional Experiment Results . . . 162
E.3.1 More Detailed Results from Section 7.5.1 . . . 162

Acknowledgments

When I reflect back on all these years as a PhD student, I find myself agreeing to what David Culler told me when I first came to Berkeley during visit days: "you will learn more during your years at Berkeley than ever before." This is so true for me. Along so many dimension, my PhD experience has been a transformative one. In the acknowledgments to follow, I will do my best to explain why I owe so many people a great debt. As with any acknowledgments, however, there is only so much that I can write. If you are reading this after the fact and wish that I had written more about you, please let me know, and I will treat you to some sugar-free boba tea or keto-friendly coffee, depending on your preferred beverage.

For a variety of reasons, I had one of the more unusual PhD experiences. However, like perhaps many students, my PhD life first felt like a struggle but over time became a highly fulfilling endeavor.

When I arrived at Berkeley, I started working with John Canny. When I think of John, the following phrase comes to mind: "jack of all trades." This is often paired with the somewhat pejorative "master of none" statement, but a more accurate conclusion for John would be "master of *all*." John has done research in a wider variety of areas than is typical: robotics, computer vision, theory of computation, computational geometry, human computer interaction, and he has taught courses in operating systems, combinatorics, and social justice. When I came to Berkeley, John had already transitioned to machine learning. I have benefited tremendously from his advice throughout the years, first primarily on machine learning toolkits when we were working on BIDMach, a library for high throughput algorithms. (I still don't know how John, a highly senior faculty, had the time and expertise to implement state-of-the-art machine learning algorithms with Scala and CUDA code.) Next, I got advice from John for my work in deep imitation learning and deep reinforcement learning, and John was able to provide technical advice for these rapidly emerging fields. As will be highlighted later, other members of his group work in areas as diverse as computer vision for autonomous driving, video captioning, natural language processing, generating sketches using deep learning, and protein folding — it sometimes seems as if all areas of Artificial Intelligence (and many areas of Human Computer Interaction) are or were represented in his group.

A good rule of thumb about John can be shown by the act of asking for paper feedback. If I ask an undergrad, I expect them to point out minor typos. If I ask a graduate student, I expect minor questions about why I did not perform some small experiment. But if I ask John for feedback, he will quickly identify the key method in the paper — and its weaknesses. His advice also extended to giving presentations. In my first paper under his primary supervision, which we presented at the Conference on Uncertainty in Artificial Intelligence (UAI) in Sydney, Australia, I was surprised to see him making the long trip to attend the conference, as I had not known he was coming. Before I gave my 20-minute talk on our paper, he sat down with me in the International Convention Centre Sydney to go through the slides carefully. I am happy to contribute one thing: that right after I

was handed the award for "Honorable Mention for Best Student Paper" from the conference chairs, I managed to get the room of 100-ish people to then give a round of applause to John. In addition, John is helpful in fund-raising and supplying the necessary compute to his students. Towards the end of my PhD, when he served as the computer science division department chair, he provided assistance in helping me secure accommodations such as sign language interpreters for academic conferences.

I also was fortunate to work with Ken Goldberg, who would become a co-advisor and who helped me transition into a full-time roboticist. Ken is a highly energetic professor who, despite being a senior faculty with so many things demanding of his time, is able to give some of the most detailed paper feedback that I have seen. When we were doing serious paper writing to meet a deadline, I would constantly refresh my email to see Ken's latest comments, written using Notability on his iPad, and then immediately rush to address them. After he surprised me by generously giving me an iPad midway through my PhD, the first thing I thought of doing was to provide paper feedback using his style and to match his level of detail in the process. Ken also provides extremely detailed feedback on our research talks and presentations, an invaluable skill given the need to communicate effectively.

Ken's lab, called the "AUTOLab," was welcoming to me when I first joined. The Monday evening lab meetings are structured so that different lab members present on research progress in progress while we all enjoy good food. Such meetings were one of the highlights of my weeks at Berkeley, as were the regular lab celebrations to his house. I also appreciate Ken's assistance in networking across the robotics research community at various conferences, which has helped me feel more involved in the research community and also became the source for my collaboration with Honda and Google throughout my PhD. Ken is very active in vouching for his students and, like John, is able to supply the compute we need to do compute-intensive robot learning research. Ken was also helpful in securing academic accommodations at Berkeley and in international robotics conferences. Much of my recent, and hopefully future, research is based on what I have learned from being in Ken's lab and interacting with his students.

To John and Ken, I know I was not the easiest student to advise, and I deeply appreciate their willingness to stick with me over all these years. I hope that the end, I was able to show my own worth as a researcher. In academic circles, I am told that professors are sometimes judged based on what their students do, so I hope that I will be able to continue working on impactful research while confidently acting as a representative example for your academic descendants.

During my first week of work at Berkeley, I arrived to my desk in Soda Hall, and in the opposite corner of the shared office of six desks, I saw Biye Jiang hunched over his laptop working. We said "hi," but this turned out to be the start of a long-time friendship with Biye. It resonated with me when I told him that because of my deafness, I found it hard to communicate with others in a large group setting with lots of background noise, and he said he sometimes felt the same but for a different reason, as an international student from China. I would speak regularly with him for four years, discussing various topics over frequent lunches and dinners, ranging from research and then to other topics such as life in

China. After he left to go to work for Alibaba in Beijing, China, he gave me a hand-written note saying: "Don't just work harder, but also live better! Enjoy your life! Good luck ^_^" I know I am probably failing at this, but it is on my agenda!

Another person I spoke to in my early days at Berkeley was Pablo Paredes, who was among the older (if not the oldest!) PhD students at Berkeley. He taught me how to manage as a beginning PhD student, and gave me psychological advice when I felt like I was hitting research roadblocks. Others who I spoke with from working with John include Haoyu Chen and Xinlei Pan, both of whom would play a major role in me getting my first paper under John's primary supervision, which I had the good fortunate to present at UAI 2017 in Sydney, Australia. With Xinlei, I also got the opportunity to help him for his 2019 ICRA paper on robust reinforcement learning, and was honored to give the presentation for the paper in Montreal. My enthusiasm was somewhat tempered by how difficult it was for Xinlei to get visas to travel to other countries, and it was partly his own experience that I recognized how difficult it could be for an international student in the United States, and that I would try to make the situation easier for them. I am also honored that Haoyu later gave a referral for me to interview at Waymo.

In November of 2015, when I had hit a rough patch in my research and felt like I had let everyone down, Florian Pokorny and Jeff Mahler were the first two members of Ken Goldberg's lab that I got to speak to, and they helped me to get my first (Berkeley) paper, on learning-based approaches for robotics. Their collaboration became my route to robotics, I am forever grateful that they were willing to work to me when it seemed like I might have little to offer. In Ken's lab, I would later get to talk with Animesh Garg, Sanjay Krishnan, Michael Laskey, and Steve McKinley. With Animesh and Steve, I only wish I could have joined the lab earlier so that I could have collaborated with them more often. Near the end of Animesh's time as a PhD student, he approached me after a lab meeting. He had read a blog post of mine and told me that I should have hung out with him more often — and I agree, I wish I did. I was honored when Animesh, now a rising star faculty at the University of Toronto, offered for me to apply for a postdoc with him. Once COVID-19 travel restrictions ease up, I promise that I will make the trip to Toronto to see Animesh, and similarly, to go to Sweden to see Florian.

Among those who I initially worked with in the AUTOLab, I want to particularly acknowledge Jeff Mahler's help with all things related to grasping; Jeff is one of the leading minds in robotic manipulation, and his Dex-Net project is one of the AUTOLab's most impactful projects, and shows the benefit of using a hybrid analytic and learned model in an age when so many have turned to pure learning. I look forward to seeing what his startup, Ambi Robotics, is able to do. I also acknowledge Sanjay's patience with me when I started working with the lab's surgical robot, the da Vinci Research Kit (dVRK). Sanjay was effectively operating like a faculty at that time, and had a deep knowledge of the literature going on in machine learning and robotics, and even databases (which was technically his original background and possibly his "official" research area, but as Ken said, "he's one of the few people who can do both databases and robotics"). His patience when I asked him questions was invaluable, and I often start research conversations by thinking about how Sanjay would

approach the question. With Michael Laskey, I acknowledge his help in getting me started with the Human Support Robot and with imitation learning. The bed-making project that I took over with him would mark the start of a series of fruitful research papers on deformable object manipulation. Ah, those days of 2017 and 2018 were sweet, while Jeff, Michael, and Sanjay were all in the lab. Looking back, there were times on Fridays when I most looked forward to our lab "happy hours" in Etcheverry Hall. Rumor has it that we could get reimbursed by Ken for these purchases of corn chips, salsa, and beer, but I never bothered. I would be willing to pay far more to have these meetings happen again.

After Jeff, Michael, and Sanjay, came the next generation of PhD students and postdocs. I enjoyed my conversations with Michael Danielczuk, who helped to continue much of the Dex-Net and YuMi-related projects after Jeff Mahler's graduation. I will also need to make sure I never stop running so that I can inch closer and closer to his half-marathon and marathon times. I also enjoyed my conversations with Carolyn Matl and Matthew Matl, over various lab meetings and dinners, about research. I admire Carolyn's research trajectory and her work on manipulating granular media and dough manipulation, and I look forward to seeing Matthew's leadership at Ambi Robotics, and I hope we shall have more Japanese burger dinners in the future.

With Roy Fox, we talked about some of the most interesting topics in generative modeling and imitation learning. There was a time in summer 2017 in our lab when the thing I looked forward to the most was a meeting with Roy to check that my code implementations were correct. Alas, we did not get a new paper from our ideas, but I still enjoyed the conversations, and I look forward to reading about his current and future accomplishments at UC Irvine. With our other postdoc from Israel, Ron Berenstein, I enjoyed our collaboration on the robotic bed-making project, which may have marked the turning point of my PhD experience, and I appreciate him reminding me that "your time is valuable" and that I should be wisely utilizing my time to work on important research.

Along with Roy and Ron, Ken continued to show his top ability in recruiting more talented postdocs to his lab. Among those who I was fortunate to meet include Ajay Kumar Tanwani, Jeff Ichnowski, and Minho Hwang. My collaboration with Ajay started with the robot bed-making project, and continued for our IROS 2020 and RSS 2020 fabric manipulation papers. Ajay has a deep knowledge of recent advances in reinforcement learning and machine learning, and played key roles in helping me frame the messaging in our papers. Jeff is an expert kinematician who understands how to perform trajectory optimization with robotics, and we desperately needed him to improve the performance of our physical robots. With Minho, I enjoyed his help on getting the da Vinci Surgical Robot back in operation and with better performance than ever before. He is certainly, as Ken Goldberg proudly announced multiple times, "the lab's secret weapon," as should no doubt be evident from the large amount of papers the AUTOLab has produced in recent years with the dVRK. I wish him the best as a faculty at DGIST. I thank him for the lovely Korean tea that he gave me after our farewell sushi dinner at Akemi's! I took a picture of the kind note Minho left to me with the box of tea, so that as with Biye's note, it is part of my permanent record. During the time these postdocs were in the lab, I also acknowledge Jingyi Xu from the Technical

University of Munich in Germany, who spent a half-year as a visiting PhD student, for her enthusiasm and creativity with robot grasping research.

To Ashwin Balakrishna and Brijen Thananjeyan, I'm not sure why you two are PhD students. You two are already at the level of faculty! If you ever want to discuss more ideas with me, please let me know. I will need to study how they operate to understand how to mentor a wide range of projects, as should be evident by the large number of AUTOLab undergraduates working with them. During the COVID-19 work-from-home period, it seemed as if one or both of them was part of all my AUTOLab meetings. I look forward to seeing their continued collaboration in safe reinforcement learning and similar topics, and maybe one day I will start picking up tennis so that running is not my only sport.

After I submitted the robot bed-making paper, I belatedly started mentoring new undergraduates in the AUTOLab. The first undergrad I worked with was Ryan Hoque, who had quickly singled me out as a potential graduate student mentor, while mentioning his interest in my blog (this is not an uncommon occurrence). He, and then later Aditya Ganapathi, were the first two undergraduates who I felt like I had mentored at least *somewhat* competently. I enjoyed working and debugging the fabric simulator we developed, which would later form the basis of much of our subsequent work published at IROS, RSS, and ICRA. I am happy that Ryan has continued his studies as a PhD student in the AUTOLab, focusing on interactive imitation learning. Regarding the fabrics-related work in the AUTOLab, I also thank the scientists at Honda Research Institute for collaborating with us: Nawid Jamali, Soshi Iba, and Katsu Yamane. I enjoyed our semi-regular meetings in Etcheverry Hall where we could go over research progress and brainstorm some of the most exciting ideas in developing a domestic home robot.

While all this was happening, I was still working with John Canny, and trying to figure out the right work balance with two advisors. Over the years, John would work with PhD students David Chan, Roshan Rao, Forrest Huang, Suhong Moon, Jinkyu Kim, and Philippe Laban, along with a talented Master's student Chen (Allen) Tang. As befitting someone like John, his students work on a wider range of research areas than is typical for a research lab. (There is no official name for John Canny's lab, so we decided to be creative and called it ... "the CannyLab.") With Jinkyu and Suhong, I learned more about explainable AI and its application for autonomous driving, and on the non-science side, I learned more about South Korea. Philippe taught me about natural language processing, summarizing text, and his "NewsLens" project resonated with me, given the wide variety of news that I read these days, and I enjoyed the backstory for why he was originally motivated to work on this. David taught me about computer vision (video captioning), Roshan taught me about proteins, and Forrest taught me about sketching. Philippe, David, Roshan, and Forrest also helped me understand Google's shiny new neural network architecture, the Transformer, as well as closely-related architectures such as OpenAI's GPT models. I also acknowledge David's help for his work getting the servers set up for the CannyLab, and for his advice in building a computer. Allen Tang's master's thesis on how to accelerate deep reinforcement learning played a key role in my final research projects.

For my whole life, I had always wondered what it was like to intern at a company like

Google, and have long watched in awe as Google churned out impressive AI research results. I had applied to Google twice earlier in my PhD, but was unable to land an internship. So, when the great Andy Zeng sent me a surprise email in late 2019, after my initial shock and disbelief wore off, I quickly responded with my interest in interning with him. After my research scientist internship under his supervision, I can confirm that the rumors are true: Andy Zeng is a fantastic intern host, and I highly recommend him. The internship in 2020 was virtual, unfortunately, but I still enjoyed the work and his frequent video calls helped to ensure that I stayed focused on producing solid research during my internship. I also appreciated the other Google researchers who I got to chat with throughout the internship: Pete Florence, Jonathan Tompson, Erwin Coumans, and Vikas Sindhwani. I have found that the general rule that others in the AUTOLab (I'm looking at you, Aditya Ganapathi) have told me is a good one to follow: "think of something, and if Pete Florence and Andy Zeng like it, it's good, and if they don't like it, don't work on it." Thank you very much for the collaboration!

The last two years of my PhD have felt like the most productive of my life. During this time, I was collaborating (virtually) with many AUTOLab members. In addition to those mentioned earlier, I want to acknowledge undergraduate Haolun (Harry) Zhang on dynamic cable manipulation, leading to the accurately-named paper *Robots of the Lost Arc*. I look forward to seeing Harry's continued achievements at Carnegie Mellon University. I was also fortunate to collaborate more closely with Huang (Raven) Huang, Vincent Lim, and many other talented newer students to Ken Goldberg's lab. Raven seems like a senior PhD student instead of just starting out, and Vincent is far more skilled than I could have imagined from a beginning undergraduate. Both have strong work ethics, and I hope that our collaboration shall one day lead to robots performing reliable lassoing and tossing. In addition, I also enjoyed my conversations with the newer postdocs to the AUTOLab, Daniel Brown and Ellen Novoseller, from whom I have learned a lot of inverse reinforcement learning and preference learning. Incoming PhD student Justin Kerr also played an enormous role in helping me work with the YuMi in my final days in the AUTOLab.

I also want to acknowledge the two undergraduates from John Canny's lab who I collaborated with the most, Mandi Zhao and Abhinav Gopal. Given the intense pressure of balancing both coursework and others, I am impressed they were willing to stick around with me while we finalized our work with John Canny. With Mandi, I hope we can continue discussing research ideas and US-China relations over WeChat, and with Abhinav, I hope we can pursue more research ideas in offline reinforcement learning.

Besides those who directly worked with me, my experience at Berkeley was enriched by the various people from other labs who I got to interact with somewhat regularly. Largely through Biye, I got to know a fair amount of Chinese international students, among them include Hezheng Yin, Xuaner (Cecilia) Zhang, Qijing (Jenny) Huang, and Isla Yang. I enjoyed our conversations over dinners and I hope they enjoyed my cooking of salmon and panna cotta. I look forward to the next chapter in all of our lives. It's largely because of my interaction with them that I decided I would do my best to learn more about anything related to China, which explains book after book that I have on my iBooks app.

My education at Berkeley benefited a great deal from what other faculty taught me during courses, research meetings, and otherwise. I was fortunate to take classes from Pieter Abbeel, Anca Dragan, Daniel Klein, Jitendra Malik, Will Fithian, Benjamin Recht, and Michael I. Jordan. I also took the initial iteration of Deep Reinforcement Learning (RL), back when John Schulman taught it, and I thank John for kindly responding to questions I had regarding Deep RL. Among these professors, I would like to particularly acknowledge Pieter Abbeel, who has regularly served as inspiration for my research, and somehow remembers me and seems to have the time to reply to my emails even though I am not a student of his nor a direct collaborator. His online lecture notes and videos in robotics and unsupervised learning are among those that I have consulted the most.

In addition to my two formal PhD advisors, I thank Sergey Levine and Masayoshi Tomizuka for serving on my qualifying exam committee. The days leading up to that event were among the most stressful I had experienced in my life, and I thank them for taking the time to listen to my research proposal. I also enjoyed learning more about deep reinforcement learning through Sergey Levine's course and online lectures.

I also owe a great deal to the administrators at UC Berkeley. The ones who helped me the most, especially during the two times during my PhD when I felt like I had hit rock bottom (in late 2015 and early 2018), were able to offer guidance and do what the could to help me stay on track to finish my PhD. I don't know all the details about what they did behind the scenes, but thank you, to Shirley Salanio, Audrey Sillers, Angie Abbatecola, and the newer administrators to BAIR. Like Angie, I am an old timer of BAIR. I was even there when it was called Berkeley Vision and Learning Center (BVLR), before we properly re-branded the organization to become Berkeley Artificial Intelligence Research (BAIR). I also thank their help in getting the BAIR Blog[1] up and running.

My research was supported initially by a university fellowship, and then later by a six-year Graduate Fellowships for STEM Diversity (GFSD)[2] which was formerly called the National Physical Science Consortium (NPSC) Fellowship. At the time I received the fellowship, I was in the middle of feeling stuck on several research progress. I don't know precisely why they granted me the fellowship, but whatever their reasons, I am eternally grateful for the decision they made. One of the more unusual conditions of the GFSD fellowship is that recipients are to intern at the sponsoring agency, which for me was the National Security Agency (NSA). I went there for one summer in Laurel, Maryland, and got a partial peek past the curtain of the NSA. By design, the NSA is one of the most secretive United States government agencies, which makes it difficult for people to acknowledge the work they do. Being there allowed me to understand and appreciate the signals intelligence work that the NSA does on behalf of the United States. Out of my NSA contacts, I would like to particularly mention Grant Wagner and Arthur Drisko.

While initially apprehensive about Berkeley, I have now come to accept it for some of the best it has to offer. I will be thankful of the many cafes I spent time in around the city,

[1] `https://bair.berkeley.edu/blog/`

[2] `https://stemfellowships.org/`

along with the frequent running trails both on the streets and in the hills. I only wish that other areas of the country offered this many food and running options.

Alas, all things must come to an end. While my PhD itself is coming to a close, I look forward to working with my future supervisor, David Held, in my next position at Carnegie Mellon University. Throughout the time when I was searching for a postdoc, I thank other faculty who took the time out of their insanely busy schedules to engage with me and to offer research advice: Shuran Song of Columbia, Jeannette Bohg of Stanford, and Alberto Rodriguez of MIT. I am forever in awe of their research contributions, and I hope that I will be able to achieve a fraction of what they have done in their careers.

In a past life, I was an undergraduate at Williams College in rural Massachusetts, which boasts an average undergraduate student body of about 2000 students. When I arrived at campus on that fall day in 2010, I was clueless about computer science and how research worked in general. Looking back, Williams must have done a better job preparing me for the PhD than I expected. Among the professors there, I owe perhaps the most to my undergraduate thesis advisor, Andrea Danyluk, as well as the other Williams CS faculty who taught me at that time: Brent Heeringa, Morgan McGuire, Jeannie Albrecht, Duane Bailey, and Stephen Freund. I will do my best to represent our department in the research area, and I hope that the professors are happy with how my graduate trajectory has taken place. One day, I shall return in person to give a research talk, and will be able to (in the words of Duane Bailey) show off my shiny new degree. I also majored in math, and I similarly learned a tremendous amount from my first math professor, Colin Adams, who emailed me right after my final exam urging me to major in math. I also appreciate other professors who have left a lasting impression on me: Steven Miller, Mihai Stoiciu, Richard De Veaux, and Qing (Wendy) Wang. I appreciate their patience during my frequent visits to their office hours.

During my undergraduate years, I was extremely fortunate to benefit from two Research Experiences for Undergraduates (REUs), the first at Bard College with Rebecca Thomas and Sven Andersen, and the second at the University of North Carolina at Greensboro, with Francine Blanchet-Sadri. I thank the professors for offering to work with me. As with the Williams professors, I don't think any of my REU advisors had anticipated that they would be helping to train a future roboticist. I hope they enjoyed working with me just as much as I enjoyed working with them. To everyone from those REUs, I am still thinking of all of you and wish you luck wherever you are.

I owe a great debt to Richard Ladner of the University of Washington, who helped me break into computer science. He and Rob Roth used to run a program called the "Summer Academy for Advancing Deaf and Hard of Hearing in Computing." I attended one of the iterations of this program, and it exposed to me what it might have been like to be a graduate student. Near the end of the program, I spoke with Richard one-on-one, and asked him detailed questions about what he thought of my applying to PhD programs. I remember him expressing enthusiasm, but also some reservation: "do you know how hard it is to get in a top PhD program?" he cautioned me. I thanked him for taking the time out of his busy schedule to give me advice. In the upcoming years, I always remembered to work hard in

the hopes of achieving a PhD. (The next time I visited the University of Washington, years later, I raced to Richard Ladner's office the minute I could.) Also, as a fun little history note, when I was there that I decided to start my (semi-famous?) personal blog,[3] which seemingly everyone at Berkeley's EECS department has seen, in large part because I felt like I needed to write about computer science in order to understand it better. I still feel that way today, and I hope I can continue writing.

Finally, I would like to thank my family for helping me persevere throughout the PhD. It is impossible for me to adequately put in words how much they helped me survive. My frequent video calls with family members helped me to stay positive during the most stressful days of my PhD, and they have always been interested in the work that I do and anything else I might want to talk about. Thank you.

[3] `https://danieltakeshi.github.io/`

Chapter 1

Introduction

Robots are widely used throughout modern society, but the vast majority of commercial robots rely on either highly predictable environments or persistent human supervision. With no environment uncertainty, robots in manufacturing can be scripted to perform a task such as spot welding or assembly with extremely high precision and repeatability [70]. In domains such as robot surgery, a human surgeon provides complete supervision by teleoperating the robot [231]. Outside of such settings, it remains an open question as to whether robots may be autonomously deployed in the real world while maintaining reliability, but successful approaches could have tremendous impact. For example, robots that can reliably manipulate arbitrary objects may address bin-picking and recycling in warehouses, cleaning an environmental disaster site, or caring for an aging population.

In order to turn this vision of robotics into a reality, robots will need to manipulate a wide variety of items. This leads to several desiderata for designing robot systems. First, in robotic manipulation [110], it is often assumed implicitly or explicitly that items are *rigid*, or can be reasonably approximated as rigid. However, many items that we humans interact with on a daily basis are highly deformable. Second, even if a system is able to manipulate a particular object, or a set of objects, the real world presents an infinite supply of new configurations and new data, so robots must maintain reliability when faced with a novel environment.

1.1 Deformable Object Manipulation

An object's configuration is a specification of the positions of all its points [131]. In this dissertation, we use the term "deformable objects" to refer to objects whose configurations cannot be sufficiently represented with a single 6 Degrees-of-Freedom (DoF) pose. This covers a wide range of objects that humans frequently interact with, such as strings, ropes, cables, clothing, towels, dough, bags, and liquids. We sometimes add the prefix "highly" to emphasize the distinction between these items and others that are technically deformable but which — depending on the application — might be adequately approximated with a

6-DoF pose, such as a stiff sponge.

While the robotics research community has driven recent advances that enable robots to grasp a wide range of rigid objects [64, 133, 135, 96, 119, 152, 153], comparatively less research has been devoted to developing algorithms that can handle deformable objects. One of the challenges in deformable object manipulation is that, as hinted earlier, it is difficult to specify such an object's configuration. For example, with a rigid cube, knowing the configuration of a fixed point relative to its center is sufficient to describe the cube's arrangement in 3D space, but a single point on a piece of fabric can remain fixed while other parts shift. This makes it difficult for perception algorithms to describe a sufficiently expressive "representation" of the fabric, especially under occlusions. Here, and in the rest of the dissertation, the term "representation" refers to the way the input is provided to an algorithm or machine learning model (*e.g.,* a deep neural network policy), and we treat an object's configuration as a special case of a possible representation.

In addition, even if one has a sufficiently descriptive representation of a deformable object, its dynamics are complex in the sense that it is difficult to predict the future configuration or representation of the deformable object after some action is applied to it. This is often needed for multi-step planning algorithms and frameworks, such as Visual Foresight [38] which uses knowledge of the dynamics to pick the best action sequence among a set of candidate action sequences.

Historically, researchers have developed ways to manipulate highly deformable objects that typically use heuristics or geometric methods. For example, Kita et al. [104, 105] and Maitin-Shepard et al. [136] rely on a heuristic for bimanual fabric manipulation where one arm lifts a fabric entirely in midair, while the second arm grasps the lowest hanging corner, detected via corner detection or other geometric techniques. These approaches can have limited generalization and may require assumptions of the fabric geometry. This thus motivates the question of whether it is possible to merge recent advances in machine learning (*i.e.,* deep learning, as we briefly review in Section 1.2) with deformable object manipulation to reduce assumptions on fabric geometry and to facilitate better generalization.

1.2 Deep Learning for Robotics

Deep learning is a branch of machine learning where the focus is on the design, evaluation, and training of functions referred to as *neural networks* [65]. Typically, the intent is to approximate arbitrarily complex functions by training on large datasets, with the model parameters optimized via backpropagation [172]. In the last decade, there have been tremendous empirical advances of deep learning for improving image classification [109, 76], semantic segmentation [130], generative modeling [66, 102], and machine translation [200, 210].

Among the many areas of Artificial Intelligence [173], one of the defining features of robotics is that it is *interactive*. Abstractly, we assume that a robot operates in an *environment*, where at a given *time step*, the information from the environment is summarized in a *state*. Given the current state, a robot performs an *action*, which influences the distribution

of subsequent states. The robot then repeats this process until a sufficient termination criteria. This applies for both simulated and real settings, and in both cases we use the term "robot" to refer to the decision-making agent. These interactive settings may be instantiated with widely-used frameworks such as imitation learning (IL) [154] and reinforcement learning (RL) [201]. In these interactive settings, a key feature of the data is that the state information from consecutive time steps will not be independent and identically distributed, breaking a common data assumption in standard machine learning analysis [19]. In addition, these settings might result in complexities such as compounding imitation errors when learning from fixed demonstration data [170] or bootstrapping errors when learning value functions [114]. Despite these challenges, deep learning has shown impressive empirical results in learning directly from images or state information in interactive settings, such as in Atari video games [144] or the game of Go [194].

A second notable feature of robotics is that the ultimate objective is typically to use robots in the *real world*, and this requires practitioners to be cognizant of issues such as imprecision of motion, hardware damage, and safety of nearby human bystanders. The robotics research community has shown potential for physical robots to learn tasks through learning-based methods in controlled laboratory settings [119, 135, 96], but many challenges remain before we can deploy autonomous robots in the unstructured real world. In addition such systems can be massively expensive in terms of physical hardware, sometimes requiring months of continual data collection across a fleet of expensive robots [120].

Given the wide applicability of deep learning in areas such as computer vision, natural language processing, and recent results in robotics, neural networks have been shown to be highly agnostic to the type of input. We therefore hypothesize that deep learning is a useful tool for robotic manipulation, particularly in the area of deformable object manipulation where a robot might reason directly with images of such objects to circumvent difficulties with state estimation.

1.3 Contributions of this Thesis

In this thesis, the objectives are to advance the frontiers of robot learning. The main robotics application is manipulation of highly deformable objects such as fabrics. We provide evidence that deep learning for processing images of deformables can lead to more robust and reliable policies compared to alternative, non-learning based methods.

This thesis presents the following technical contributions:

- We introduce and formalize new tasks for robotic manipulation of highly deformable objects, which include bed-making, wrapping items, and manipulating bags via opening bags, inserting item(s) in them, and transporting bags to target zones.
- We formalize novel model-free and model-based learning approaches for fabric smoothing and folding, which have been investigated in prior research. We compare the proposed approaches to non-learning approaches based on heuristics or geometric features.

- We implement and release an open-source fabric simulator designed for fabric smoothing and folding, where the fabric itself is implemented as an $N \times N$ grid of vertices, with forces between pairs of vertices.
- Using the fabric simulator, we show simulation-to-real transfer by learning fabric smoothing and folding in simulation, where perfect information about the fabric is available for either training a model of the fabric physics or by utilizing an algorithmic supervisor.
- We implement and release an open-source benchmark for manipulation of highly deformable objects built on PyBullet [30] for physics simulation. The benchmark contains a range of tasks involving 1D deformables (cables), 2D deformables (fabrics) and 3D deformables (bags), where the bags are designed by taking a 2D deformable and arranging it in a spherical-like shape so it can contain items.
- We design a novel neural network architecture, *Goal-Conditioned Transporter Networks*, which extends Transporter Networks [240] for image-based goal-conditioned object rearrangement. Hence, an image of the desired configuration of the objects is passed along with the current image to the goal-conditioned architecture.
- Across a wide range of deformable object manipulation tasks, we show the benefit of using depth images as an input modality, in addition to color images.
- We formalize the problem of learning from a set of multiple teachers, which can each provide demonstrations to an RL agent (*i.e.,* student).
- We present a framework for designing different curricula for a fixed, offline data. The framework, combined with standard off-the-shelf RL algorithms such as TD3 [57], can result in successful offline RL despite known challenges [121].
- We leverage curriculum learning for faster learning in RL from demonstrations or in offline RL. In either case, the key idea is to select a range of samples (equivalently, teachers, as they provide the data) that are at a suitable difficulty level for the student. We test these on standard Atari [15] and MuJoCo [208] benchmarks.

Many of the above contributions are present across multiple chapters in this thesis (see Section 1.4), highlighting the widespread applicability of these ideas.

1.4 Thesis Structure

We structure the thesis as follows:

- In Chapter 2, we present a system that can perform robot bed-making on a quarter-scale bed with one blanket. We use two mobile robots: the Toyota Human Support

Robot [72] and the Fetch Robot [221], and show the benefit for learning-based approaches. This work was previously published at ISRR 2019 [185].

- In Chapter 3, we develop a fabric simulator for smoothing, and demonstrate how to perform simulation-to-real transfer to a physical da Vinci surgical robot. By doing so, we avoid the tedious physical data collection we did for bed-making (from Chapter 2). This work was previously published at IROS 2020 [184].

- In Chapter 4, we present a closely-related project where we build on top of Chapter 3 to perform model-based reinforcement learning for goal-conditioned fabric manipulation. This enables a single policy to be used for a variety of smoothing and folding tasks. This work was previously published at RSS 2020 [85], and was later extended into a journal paper in *Autonomous Robots* [86].

- In Chapter 5, we propose (1) a benchmark for deformable object manipulation by creating a series of open-source PyBullet [30] tasks involving cables, fabrics, and bags, and (2) a novel goal-conditioned architecture for manipulation of deformables, called Goal-Conditioned Transporter Networks. This work was previously published at ICRA 2021 [187].

- In Chapter 6, we switch to *curriculum learning*, or how to design a student's learning process so that it learns at appropriate "difficulty levels" [39, 16]. We study when a student has access to multiple teachers with different levels of expertise, and we advocate for picking teachers who can provide samples at an appropriate difficulty level. This work was previously presented at the 2019 NeurIPS Deep RL workshop [188].

- In Chapter 7, we continue studying curriculum learning in the context of continuous control and offline reinforcement learning. We show that restricting the range of samples an agent can learn from in a replay buffer, and varying this throughout the learning process, can result in stable offline learning. This work is currently undergoing peer-review.

- In Chapter 8, we recap what we have learned from the thesis, and discuss some of the most promising ideas for future work.

All the chapters that contain published research results have supplementary material and code available. Source code will typically be available on GitHub.[1]

[1]My GitHub profile is `https://github.com/DanielTakeshi` which will also link to my most up to date academic website.

Part I

Case Study: Robot Bed-Making

Chapter 2

Deep Transfer Learning of Pick Points on Fabric for Robot Bed-Making

Fabric manipulation remains challenging for robots, with real-world applications ranging from folding clothing to handling tissues and gauzes in robotic surgery. In contrast to rigid objects, fabrics have infinite dimensional configuration spaces. In this work, we focus on computing suitable *pick points* for blankets that facilitate smoothing and coverage.

Designing an analytic model is challenging because a blanket is a complex manifold that may be highly deformed, wrinkled, or folded. We present an approach based on deep learning to find pick points from exposed blanket corners, where the blankets are strewn across a flat surface (Figure 2.1). We use depth images as the input modality for invariance to different colors and patterns (Figure 2.2).

Consider bed-making with a mobile manipulator. Bed-making is a common home task which is rarely enjoyed and can be physically challenging to senior citizens and people with limited dexterity. Surveys of older adults in the United States [14, 45], suggest that they are willing to have a robot assistant in their homes, particularly for physically demanding tasks. Bed-making is well-suited for home robots since it is tolerant to error and not time-critical [20, 47]. While prior work has designed robotic beds equipped with pressure sensors to anticipate patient pose [190] or to lift people in and out of bed [146], we apply deep transfer learning to train a single-armed mobile robot with depth sensors to cover a bed with a blanket, without relying on sophisticated bed-related sensors or mechanical features.

The contributions of this chapter include: (1) a deep transfer learning approach to selecting pick points that generalizes across robots and blankets, and (2) a formalization of robot bed-making based on pick points and maximizing blanket coverage. We present experimental data from two robots, three blankets, and three pick point methods, demonstrating that learned pick points can achieve coverage comparable to humans. Code and data are available at `https://sites.google.com/view/bed-make`.

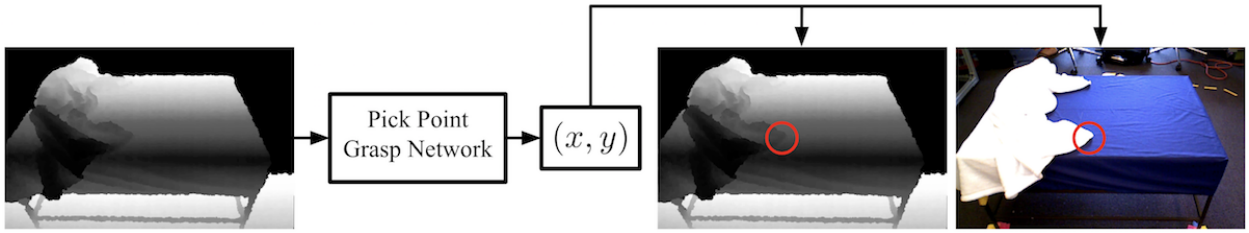

Figure 2.1: System overview: the depth image from the robot's head camera sensors is passed as input to the grasp neural network. The output is the pixel-space pick point (x, y), marked by the red circle in the depth and RGB image pair. The robot grasps at the blanket location corresponding to the pick point, and pulls it towards the nearest uncovered bed surface corner (to the lower right in this example).

2.1 Related Work

Fabric manipulation has been explored in a variety of contexts, such as in assistive and home robotics through sewing [182], ironing [124], dressing assistance [100, 41], and folding of towels and laundry [143, 191].

Early research in manipulating fabric used a dual-armed manipulator to hold the fabrics in midair with one gripper, using gravity to help expose borders and corners for the robot's second gripper. Osawa et al. [155] proposed the pick point technique of iteratively re-grasping the lowest hanging point as a subroutine to flatten and classify clothing. Kita et al. [104, 105] used a deformable object model to simulate hung clothing, allowing the second gripper to grasp at a desired point.

Follow-up work generalized to manipulating unseen articles of clothing and handling a wider variety of initial cloth configurations. For example, Maitin-Shepard et al. [136] identified and tensioned corners to enable a home robot to fold laundry. Their robot grasped the laundry fabric in midair and rotated it to obtain a sequence of images, and used a RANSAC [50] algorithm to fit corners to cloth borders. The robot then gripped any corner with its second gripper, let the cloth settle and hang, and gripped an adjacent corner with its original gripper. Cusumano-Towner et al. [31] followed-up by improving the subroutine of bringing clothing into an arbitrary position. They proposed a hidden Markov model and deformable object simulator along with a pick-point strategy of regrasping the lowest hanging point. Doumanoglou et al. [37] extended the results by using random forests to learn a garment-specific pick point for folding. These preceding papers rely on gripping the fabrics in midair with a dual-armed robot. In contrast, we focus on finding pick points on fabric strewn across a horizontal surface. Furthermore, the fabrics we use are too large for most dual armed robots to grip while also exposing a fabric corner in midair.

Recently, pick points for fabric manipulation have been learned via reinforcement learning in simulation and then conducting sim-to-real transfer. Thananjeyan et al. [205] developed

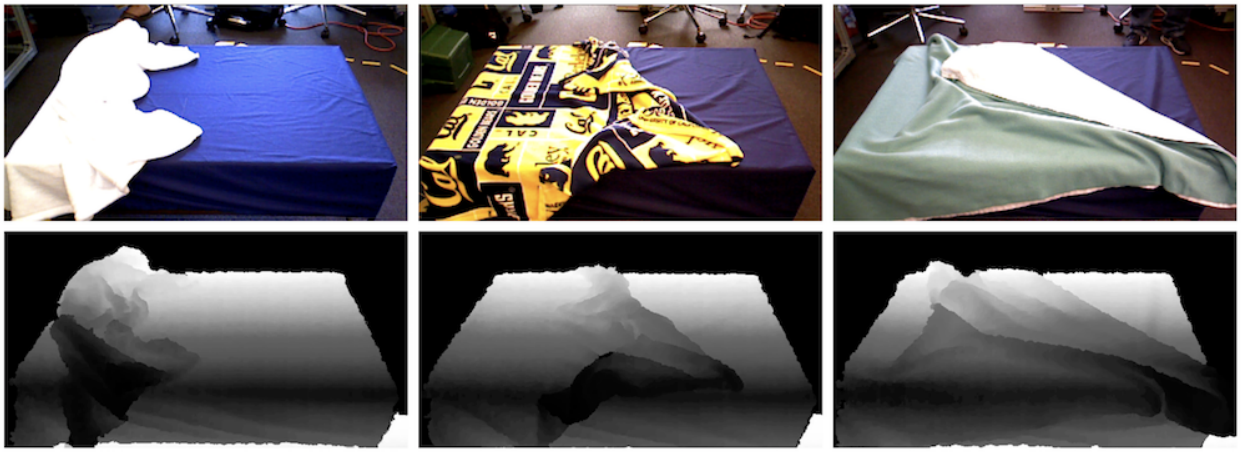

Figure 2.2: Quarter-scale bed with blue bottom sheet and three different blankets: white (left), multicolored yellow and blue (Y&B) pattern (middle), and teal (right). The system is trained on depth images (bottom row).

a tensioning policy for a dual-armed surgical robot to cut gauze. One arm pinched the gauze at a pick point and pulled it slightly. The resulting tension made it easier for the second arm, with a scissor, to cut the gauze. Pick points were selected by uniformly sampling candidates on the gauze and identifying the best one via brute force evaluation. Matas et al. [139] benchmarked a variety of deep reinforcement learning policies for grasping cloth in simulation, and showed transfer to physical folding and hanging tasks. Their policy outputs a four dimensional action, where the first three are gripper velocities and the last one represents the opening or closing of the gripper for finalizing pick points. Running pure reinforcement learning on physical robots remains difficult due to wear and tear. Thus, these approaches for learning pick points often rely on having accurate environment simulators, but these are challenging to design and not generally available off-the-shelf for robotics tasks.

Additional fabric manipulation techniques include trajectory transfer and learning from image-based wrinkles. Schulman et al. [183] propose to adapt a demonstrated trajectory (including pick points) from training time to the geometry at test time by computing a smooth, nonrigid transformation. This technique, however, assumes that a nonrigid transformation is possible between two point clouds of the deformable object, which is not generally the case for two different blanket setups. Jia et al. [94] present a cloth manipulation technique which learns from image histograms of wrinkles. While they showed impressive human-robot collaboration results in flattening, folding, and twisting tasks, their dual-armed robot grasped two cloth corners at initialization and kept the grippers closed while moving the arms. Thus, the pick point decision is pre-determined, whereas we learn pick points using deep learning on depth maps.

2.2 Methodology

Problem Statement We assume a mobile robot with an arm, a gripper, and color and depth cameras. We assume the robot can position itself to reach anywhere on the half of the bed closest to its position. Let $\pi_\theta : \mathbb{R}^{640\times480} \rightarrow \mathbb{R}^2$ be a function parameterized by θ that maps a depth image $\mathbf{o}_t \in \mathbb{R}^{640\times480}$ at time t to a pick point $\mathbf{u}_t = \pi_\theta(\mathbf{o}_t)$ for the robot. We are interested in learning the parameters θ such that the robot grasps at the pick point and pulls the blanket to the uncovered bed frame corner nearest to it.

We represent the resulting blanket configuration with an occupancy function $\xi : \mathbb{R}^3 \rightarrow \{0, 1\}$ to determine if a point is part of the blanket or not. Let $c : \xi \rightarrow \mathbb{R}$ be a function representing a desired performance metric. We define $c(\xi)$ as *blanket coverage*, and measure it from a top-down camera as the percentage of the top bed plane covered by blanket ξ.

2.2.1 Setup

Figure 2.1 shows an overview of the system. A depth image is presented to a grasp network (described in Section 2.2.2) which estimates the location of a suitable pick point. The robot then moves its gripper to the location, closes it, and pulls in the direction of the nearest uncovered corner of the bed frame. Due to the stochastic nature of the task, the robot may not be able to achieve sufficient coverage in a single attempt. For example, the blanket may slip out of the robot's grip during pulling. The robot uses a bed coverage heuristic to decide if it should attempt another grasp and pull at the same side. In this work, we limit the robot to four attempts at each of the two sides of the bed.

2.2.2 Grasp Network for Pick Points

The robot captures the depth image from its head camera sensors as *observation* $\mathbf{o}_t$. We define a *grasp network* π_θ as a deep convolutional neural network [109] that maps from observation $\mathbf{o}_t$ to a pixel position $\mathbf{u}_t = (x, y)$ where the robot will grasp (i.e., the pick point). We project this point onto the 3D scene by first measuring the depth value, z, from the corresponding depth image. We then project (x, y, z) using known camera parameters, and set the gripper orientation to be orthogonal to the top surface of the bed.

The network π_θ is based on YOLO [168], a single shot object detection network for feature extraction. We utilize pre-trained weights optimized on Pascal VOC 2012 [44]. We call this network *YOLO Pre-Trained*, and show its architecture in Figure 2.3. We fix the first 32 million parameters from YOLO and optimize two additional convolutional layers and two dense layers.

Since YOLO Pre-Trained has weights trained on RGB images and we use *depth* images with the single channel repeated three times to match RGB dimensions, we additionally tested full training of YOLO without fixing the first 32 million parameters. This, however, converged to a pixel error twice as large.

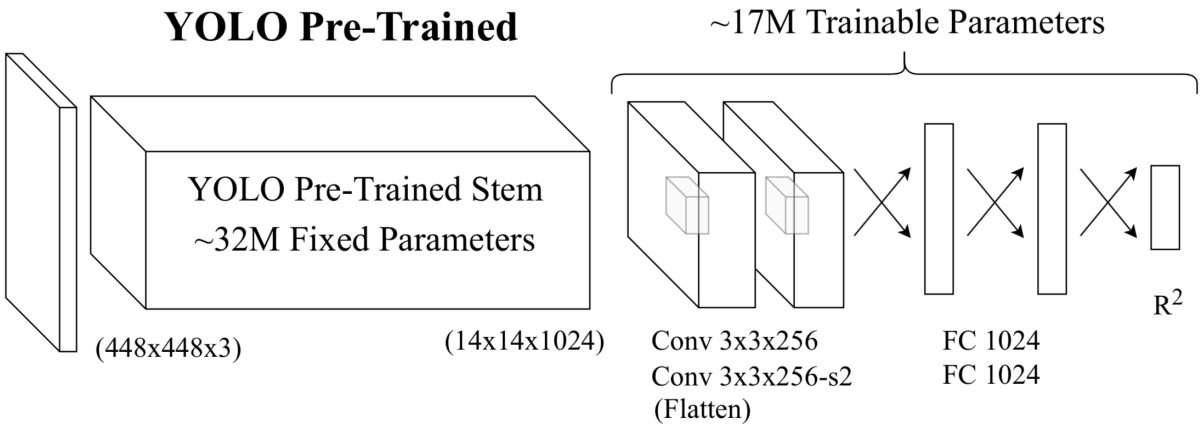

Figure 2.3: The grasp network architecture. From a $(448 \times 448 \times 3)$-sized input image, we use pre-trained weights to obtain a $(14 \times 14 \times 1024)$ tensor. The input is resized from the original (640×480) depth images. We triplicate the depth image along three channels for compatibility with pre-trained weights. The notation: "-s2" indicates a stride of two, and two crossing arrows are a dense layer.

2.3 Data and Experiments

We use two mobile manipulator robots, the HSR [72] and the Fetch [221] (see Figure 2.4) to evaluate the generality of this approach. The HSR has an omnidirectional base and a 3 DoF arm. The Fetch has a longer 7 DoF arm and a differential drive base. Both robots have PrimeSense head camera sensors with RGB and depth sensors. The robot's task is to make a quarter-scale bed with dimensions $W = 67$ cm, $H = 45$ cm, and $L = 91$ cm. The bed consists of one blanket with area slightly larger than the top surface so that a human can comfortably cover it. One end of the blanket is fixed to one of the shorter sides of the bed frame to simulate two corners being tucked under a mattress. Figure 2.4 shows a third-person view of the experimental setup with both robots.

2.3.1 Data Collection and Processing

To facilitate automatic labeling, we used a white blanket with a red mark at its corner to define the pick point. The red mark is solely used for training labels, as the grasp network does not see it in the depth images on which it is trained.

To sample initial states, the human supervisors (the first two authors) fixed a blanket end to one of the shorter edges of the bed and tossed the remaining part onto the top surface. If the nearest blanket corner was not visible or unreachable from the robot's position, the supervisors re-tossed the blanket. Figure 2.5 demonstrates examples of initial states as viewed through the robot's head camera sensors. From the initial state, the human

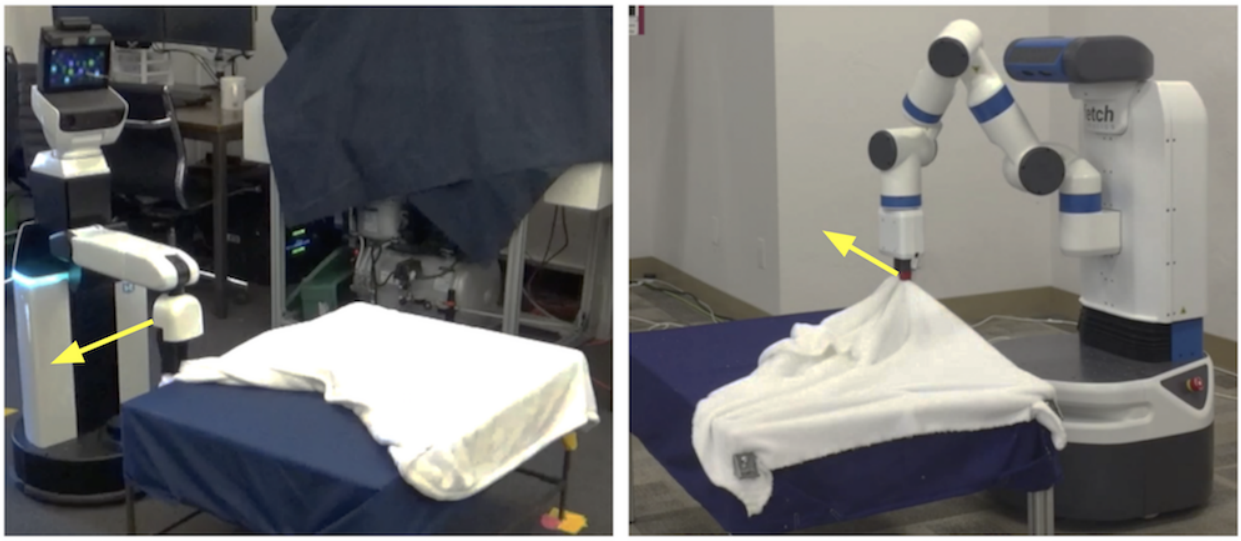

Figure 2.4: The HSR (left) and Fetch (right) performing the bed-making task with a white blanket. The goal is to cover the (dark blue) top bed surface under the blanket. The yellow arrows indicate the direction of the arm motion.

supervisors performed short pulls of the blanket and recorded the robot's depth camera image and the pick point after each pull. We took 1028 and 990 images from the camera sensors of the HSR and Fetch, respectively, which we oriented at roughly the same position to keep the viewing angle consistent. The resulting dataset $\mathcal{D} = \{(\mathbf{o}_i, \mathbf{u}_i)\}_{i=1}^{N}$ is used for training the grasp network with $N = 2018$. Labels $\mathbf{u}_i$ correspond to pixel coordinates of the red marked location in $\mathbf{o}_i$.

We performed several data pre-processing steps to better condition the optimization. To avoid noise from distant background objects, we set all depth values beyond 1.4 meters from the robot's head camera to zero. Then, depth values are scaled into the range $[0, 255]$ to form $\mathbf{o}_i$, matching the scale of pixels in the RGB images used for pre-trained weights. We also apply the following data augmentation techniques on $\mathbf{o}_i$: adding uniform noise in the range $[-4, 4]$, adding zero-mean Gaussian noise with standard deviation $\sigma = 15$, adding black dots randomly to 0.4% of the pixels, adding black or white dots randomly (again, to 0.4% of the pixels), and a flip about the vertical axis to simulate being on the opposite side of the bed. These techniques result in 10x more training data.

The parameters of π_θ are optimized via Adam [101] by minimizing the L_2 loss. The learning rate and L_2 regularization strength hyperparameters are chosen based on 10-fold cross-validation performance, where for each fold we train until the validation L_2 error stops decreasing. For deployment, we train the grasp network using the combined data from all 10 folds, on the best hyperparameters.

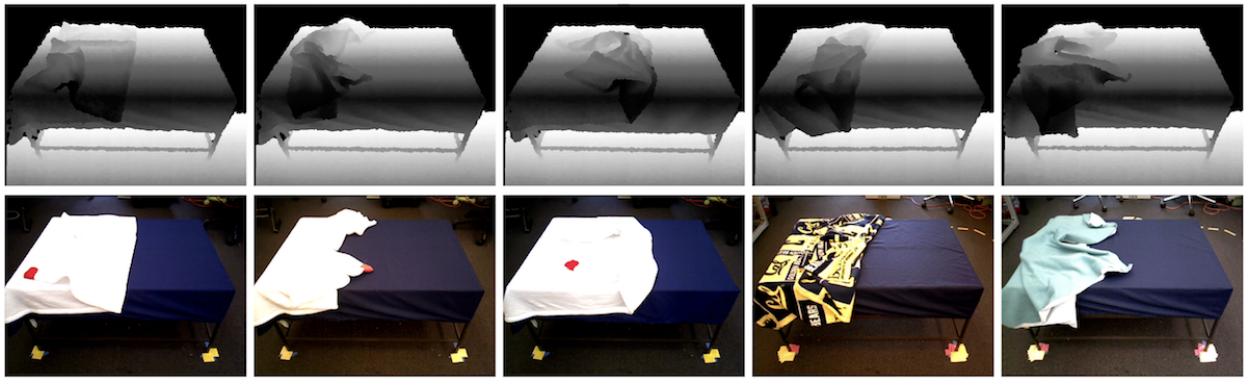

Figure 2.5: Examples of initial states. The grasp network is trained with depth images (top row). To avoid background noise, we black out regions beyond the validation-tuned depth value of 1.4 meters. We also show the corresponding RGB images (bottom row). The training data is automatically labeled with the red marker from the RGB image. During testing, a network trained on the white blanket is sometimes applied on the Y&B and the teal blankets shown in the last two columns.

2.4 Results

In this section, we present pick point prediction and blanket coverage results.

2.4.1 Grasp Network Training

To evaluate the accuracy of pick point estimation, we analyzed the L_2 error between the estimated pick point and the ground truth. Figure 2.6 (left) demonstrates training results of the grasp network over the best hyperparameter set. It shows the L_2 pixel prediction losses as a function of training epoch, indicating that it converges to 27 pixel error. This pixel error for the grasp network corresponds to the 93% and 89% coverage results for the network that we later report in Section 2.4.3 and Figure 2.9.

Figure 2.6 also presents a scatter plot of the distribution of training points (i.e., pick points) and a heat map of those points and their held-out L_2 losses in pixels for the best-performing validation set iteration. Not all the scatter plot points are on top of the bed surface shown in Figure 2.6. The scatter plot and heat map are only overlaid over a *representative* image that the robot might see during the task. Though we set the robot to be at roughly the same position each time we collect data or run the task, in practice there are variations due to imperfections in robotic base motion so the precise pixel location of the bed is not fixed. We observe that the heat map shows darker regions towards the extremes of the dataset, particularly to the left and bottom. These correspond to when blanket corners occur near the edge of the bed surface.

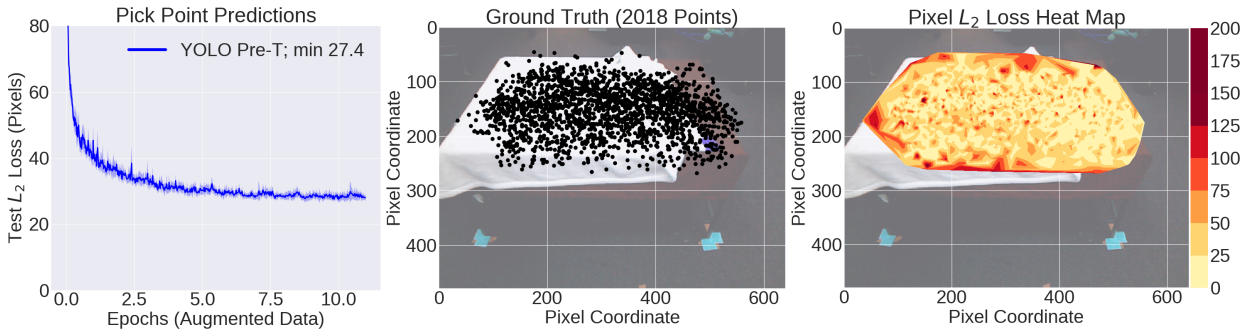

Figure 2.6: Left: validation set L_2 errors (in pixels) when training the grasp network on depth data, averaged over 10-fold cross validation with one standard deviation shaded. Middle: scatter plot showing the distribution of corners (i.e., pick points) in the combined data of 2018 images. Right: heat map of the L_2 error in pixel space. The second and third subplots are overlaid on top of a representative image of the bed to aid visualization; see Section 2.4.1 for details. We report results for the model with best validation set performance.

We next investigate the effect of training data size on L_2 pixel error of the pick points. We set aside a held-out set of 205 images, roughly the size of one fold in 10-fold cross validation. We use the remaining $2018 - 205 = 1813$ images for training, which we sampled to get 4 training datasets of increasing sizes: $\frac{1}{9}, \frac{2}{9}, \frac{4}{9}, \frac{6}{9}$ of the training dataset. Figure 2.7 plots a single training run for each of the subsets, as a function of total training points consumed. The number of points consumed is the number of Adam gradient steps multiplied by the batch size, which was fixed at 32. The results suggest a clear benefit to having more training data, with curves corresponding to larger datasets converging to lower L_2 error. Nonetheless, we observe diminishing returns at roughly two-thirds of the full training set. While the largest training size gets around 30 L_2 error, using about two-thirds of it can attain almost the same test error. For the rest of this chapter, we report results with π_θ trained on all the training data.

2.4.2 Harris Corner Detector is Insufficient

We also investigated whether classical corner detection methods such as the Harris Corner Detector (HCD) [71] can be used for selecting blanket corners as pick points. We applied the HCD to get a set of candidates. Then, we selected the corners lying on the bed surface, and picked the one closest to the bottom right of the image. We used a detector tuned for high sensitivity[1] on 202 depth images of our dataset (Section 2.3.1). Figure 2.8 shows three representative depth images with corner estimates from the grasp network, the HCD, and the ground truth. In some instances HCD fails to detect any corners, and in others,

[1]We used the OpenCV implementation with block size at 2, Sobel derivative aperture parameter at 1, and free parameter k at 0.001.

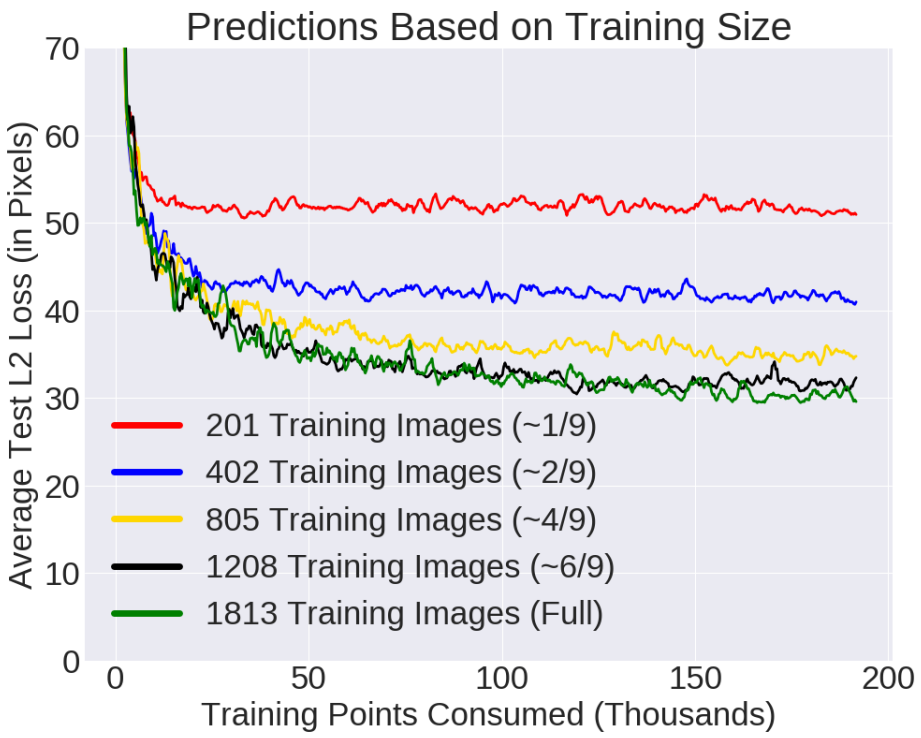

Figure 2.7: Average L_2 pixel error on a held-out set of 205 testing images for five training sets of different sizes. Data sizes range from 201 to 1813 images. Results are a function of the cumulative number of data points consumed (and not epochs, which depend on the overall data size) during training, so the curves also measure efficiency of reducing the L_2 pixel error. The results suggest diminishing returns on more data once we hit around 2/3 (i.e., 1208 images) of the largest size here.

it returns many false positives. Overall, it failed to detect corners in 20 images (10%) and, when a corner was detected, the average L_2 pixel error was 175.0, about 6x worse than the best-performing grasp network presented in Section 2.4.1. Based on these results, we do not pursue the corner detection approach in the rest of this chapter.

2.4.3 Physical Robot Deployment Evaluation

To evaluate the proposed system, we deployed the trained network on two robots. We benchmark the results versus two alternative pick point methods: an analytic highest-point baseline and a human supervisor. For each experimental condition, we report average coverage before and after each rollout. To evaluate blanket coverage, we measure the area of the bed's top surface and the area of its uncovered portion using contour detection on a top-down camera image. All images used for coverage results in the following experiments

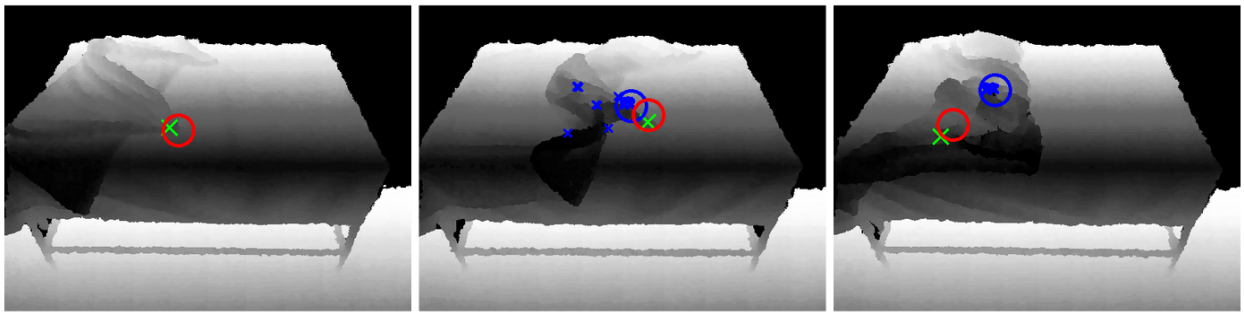

Figure 2.8: Three representative depth images, with blanket corner estimates overlaid from the Harris Corner Detector (blue "X") and the grasp network (red circle), along with the ground truth (green "X"). The blue circle represents the bottom right corner from the set of candidates found by the Harris Corner Detector. Left: no corners are detected on the top surface. Middle: the detector finds many false positives, but some corners are close to the ground truth. Right: all detected corners are false positives.

are on the project website.

Bed-Making Rollout

To start each rollout, the robot is positioned at one of the longer sides of the bed. The robot determines a pick point on the blanket based on one of three methods (analytic, human, or learned) then grasps and pulls it to the nearest uncovered bed corner. The robot repeats the process on the second side of the bed.

As stated in Section 2.2.1, the robot is allowed up to four grasp and pull attempts per side. Thus, after each attempt, the robot visually checks if its action resulted in sufficient coverage. We can use a variety of heuristics to measure coverage and encode this behavior. For these experiments, we trained a second deep neural network with the same architecture as the grasp network. We trained it to detect, given the depth image, if the side closest to the robot is sufficiently covered. This network was applied to all experimental conditions to allow us to focus on comparing pick point methods.

Initialization and Pick Point Method Selection

To ensure the evaluations are fair with respect to the initial blanket state, the blanket is tossed on the the bed frame, *then* one of the three pick point methods (analytic, human, or learned) is selected at random. This was repeated until we achieved a minimum of 24 and 15 rollouts per method for the HSR and Fetch, respectively.

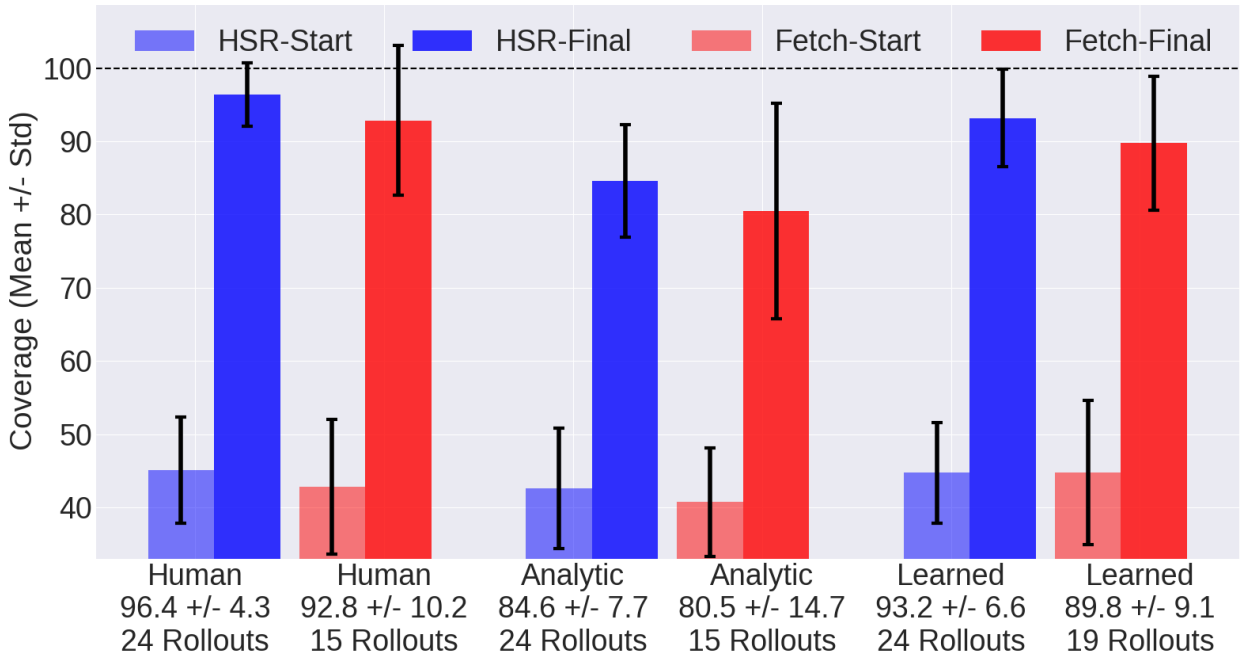

Figure 2.9: Bed coverage results with the HSR (blue) and Fetch (red) at the start and end of each rollout, all applied on a bed with the white blanket. *Human*: human selecting pick points via a click interface. *Analytic*: the highest reachable point baseline. *Learned*: depth-based neural network, the pick point method we propose in this chapter based on depth images of the blanket. The error bars represent one standard deviation.

Analytic Highest Point Baseline

We benchmark with an analytic baseline where the robot grips the highest reachable point on the blanket. Figure 2.9 shows that the analytic baseline achieves $85 \pm 8\%$ and $81 \pm 15\%$ coverage for the HSR and Fetch, over 24 and 15 rollouts, respectively.

The analytic method performs reasonably well, but has high variance. The highest point may correspond to a corner fold, in which case the analytic method will significantly increase coverage. When the pick point is not at a corner fold, the analytic method tends to use multiple grasp and pull actions, which also increases coverage. After the first attempt, however, the blanket corner frequently gets folded under, limiting future coverage increases. Figure 2.10 shows the HSR following the highest point baseline grasping procedure. After the robot grasps and pulls the highest point, it attains significant coverage, but the blanket is at a state that little further improvements can be made.

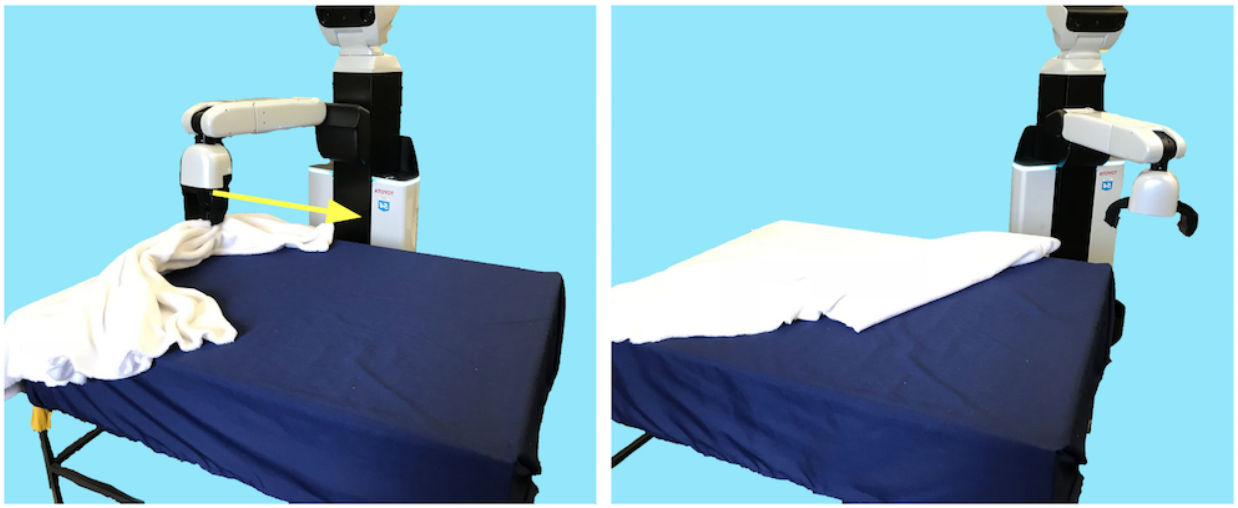

Figure 2.10: The HSR following the analytic baseline. It grasps at the highest reachable point and pulls in the direction of the yellow arrow, improving coverage.

Human Supervisor

As another benchmark, we have humans (the first two authors) select pick points via a click interface over a web-server application. The human was only provided the usual depth image as input, to be consistent with the input to the grasp network, and did not physically touch the robot or the blanket. Figure 2.9 suggests that the human supervisor achieves $96 \pm 4\%$ and $93 \pm 10\%$ coverage for the HSR and Fetch, over 24 and 15 trials, respectively.

Learned Grasp Network

Figure 2.9 shows that on the white blanket, the learned grasp network π_θ attains $93 \pm 7\%$ and $90 \pm 9\%$ coverage for the HSR and Fetch, over 24 and 19 trials, respectively (with an average coverage of 92% for both robots combined). This outperforms the analytic baseline, demonstrating the feasibility of a learning based approach. Moreover, the trained grasp network performs nearly as well as the human supervisor. These results are consistent among the two robots, providing evidence of the robot-to-robot transfer capability of the method. The accompanying video shows a rollout of the learned grasp network deployed on both robots.

To test for statistical significance among different experimental conditions, we use the Mann-Whitney U test [137]. The standard t-test is not used as the coverage metric is not normally distributed. For the HSR, the Mann-Whitney U test for the analytic versus learned network (when applied on the white blanket setup) is $p = 0.00034$, a strong indicator of statistical significance. The same test for the learned network versus a *human* results in

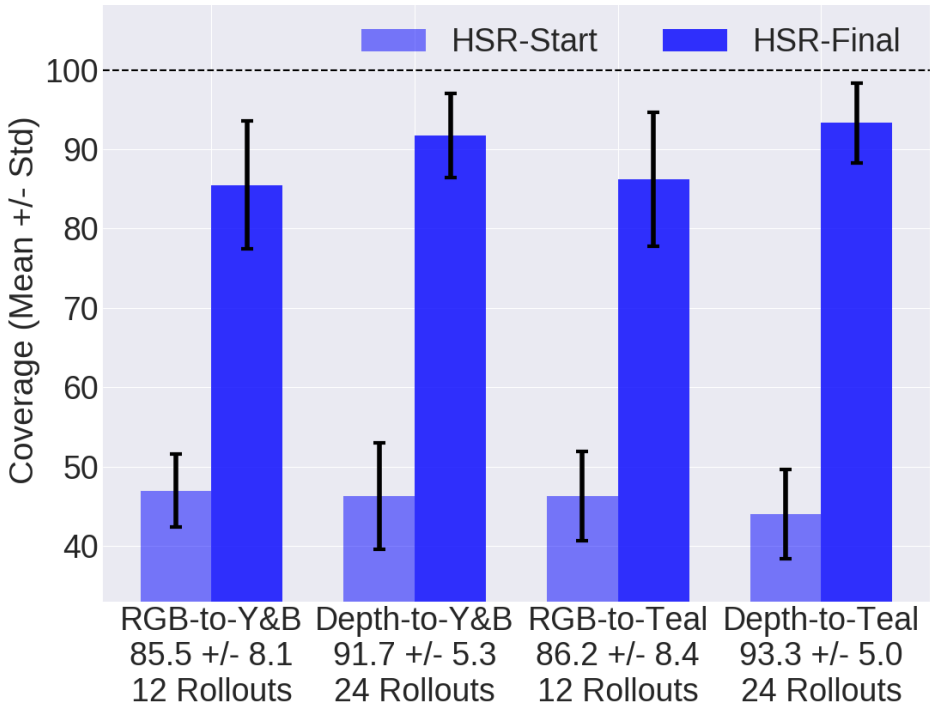

Figure 2.11: Results of generalization to blankets with different colors and patterns using the HSR. *RGB-to-Y&B* and *RGB-to-Teal*: the RGB-based grasp network on Y&B and teal blankets. *Depth-to-Y&B* and *Depth-to-Teal*: depth-based grasp network on Y&B and teal blankets. The error bars represent one standard deviation.

a higher value of $p = 0.0889$, suggesting that the learned grasp network's performance more closely matches the human supervisor than the analytic baseline.

Average Number of Grasps

We analyzed the average number of grasp attempts made by each method. Each rollout was limited to a maximum of eight grasp attempts (four per side). The highest point baseline required an average of 6.2 and 4.9 attempts for the HSR and Fetch, respectively, compared to 4.4 and 4.3 for the learned network, and 2.8 and 3.0 for the human supervisor. The baseline method used more attempts because it often covered the blanket corner during the first pull and failed to make progress in the remaining grasps before reaching the limit of four attempts per side.

Depth-Based Grasp Network Generalization

We tested the generalization capability of the method to blankets with different colors and patterns: the multicolored Y&B and teal blankets shown in Figure 5.1. We deployed the same grasp network (trained on depth images of the white blankets) on the HSR with 24 rollouts for each of the two other blankets. The results in Figure 2.11 show that the HSR attains 92±5% and 93±5% coverage for the Y&B and teal blankets, respectively. The Mann-Whitney U tests for the depth-based grasp network on white versus Y&B, white versus teal, and teal vs Y&B blankets (24 rollouts for each comparison) are $p = 0.227$, $p = 0.844$, and 0.327, respectively. These relatively high p-values mean we cannot reject a hypothesis that the coverage samples in each group are from the same distribution, suggesting that the grasp neural network trained on depth images (of the white blanket) directly transfers to two other blankets despite slightly different material properties; the Y&B blanket is thinner, while the teal blanket is less elastic and has a thin white sheet pinned underneath it.

RGB-Based Grasp Network Generalization

We trained a new grasp network using the *RGB images* of the white blanket to compare with the depth-based approach. Other than this change, the RGB-based network was trained in an identical manner as the depth-based network. From Figure 2.11, we observe that the RGB-based network obtains $86 \pm 8\%$ and $86 \pm 8\%$ coverage on the Y&B and teal blankets, suggesting that the depth-based method generalizes better to different colors and patterns. Empirically, we observe that the RGB-based network consistently grasps the Y&B blanket close to the center of its area over the bed's top surface. For the teal blanket, it tends to pick anywhere along the exposed white underside, or near the center if there is no white visible.

2.4.4 Timing Analysis

We performed experiments on a single workstation with an NVIDIA Titan Xp GPU, and list timing results in Table 2.1 for the three major components of the bed-making rollout: moving to another side of the bed, physical grasp execution, and grasp network forward passes (if applicable). The reported numbers combine all relevant trials from Figures 2.9 and 2.11, with 144 and 49 total rollouts for the HSR and Fetch, respectively. The major bottlenecks are moving to another side, which required 32 ± 2 s and 29 ± 22 s for the HSR and Fetch, respectively, and grasp execution, which took 18 ± 2 s and 88 ± 19 s. In contrast, the fast single-shot CNN required a mean time of just 0.1 ± 0.02 s.

2.5 Conclusion and Future Work

In this chapter, we presented a supervised learning approach to select effective pick points on fabrics. The method uses depth images to be invariant to the color and pattern of the fabric. We applied the method to a quarter-scale bed-making task, where the robot grasps

Table 2.1: Timing results of bed-making rollouts for the HSR and Fetch, all in seconds, and the number of times the statistic was recorded ("quantity"). *Moving to a Side*: moving from one side of the bed to another, which happens once per rollout. *Grasp Execution*: the process of the robot moving its end-effector to the workspace and pulling to a target. *Neural Network Pass*: the forward pass through the grasp network, which is not recorded for the analytic and human pick point methods.

	Component	**Mean Time (Sec.)**	**Quantity**
HSR	Moving to a Side	32 ± 2	144
	Grasp Execution	18 ± 2	706
	Neural Network Pass	0.1 ± 0.2	491
Fetch	Moving to a Side	29 ± 22	49
	Grasp Execution	88 ± 19	201
	Neural Network Pass	0.1 ± 0.2	82

at a pick point and pulls the blanket to the edge of the bed to increase coverage. We trained a grasp network to estimate pick points, and deployed it on the HSR and Fetch. Results suggest that the proposed depth-based grasp network outperforms an analytic baseline that selects the highest blanket point, and better generalizes to different fabrics as compared to an RGB-based network.

In future work, we will explore scalable representations for learning pick points with Fog robotics [203]. We will also extend the bed-making task by reducing the number of hard-coded robot actions and learning policies that can handle a wider variety of blanket configurations. Such policies may be learned using reinforcement learning in combination with cloth simulators to decrease the number of real-world examples required. Finally, we plan to explore application to manipulating furniture covers, table cloths, textiles, and other deformable objects.

Part II

Manipulating Fabrics from Simulation

Chapter 3

Deep Imitation Learning of Sequential Fabric Smoothing From an Algorithmic Supervisor

Robot manipulation of fabric has applications in senior care and dressing assistance [41, 42, 63], sewing [182], ironing [124], laundry folding [123, 143, 191, 230], fabric upholstery manufacturing [158, 209], and handling gauze in robotic surgery [205]. However, fabric manipulation is challenging due to its infinite dimensional configuration space and unknown dynamics.

We consider the task of transforming fabric from a rumpled and highly disordered starting configuration to a smooth configuration via a series of grasp and pull actions. We explore a deep imitation learning approach based on a Finite Element Method (FEM) fabric simulator with an algorithmic supervisor and use DAgger [170] to train policies. Using color and camera domain randomization [175, 207], learned policies are evaluated in simulation and in physical experiments with the da Vinci Research Kit (dVRK) surgical robot [97]. Figure 5.1 shows examples of learned smoothing episodes in simulation and the physical robot.

This chapter contributes: (1) an open-source simulation environment and dataset for evaluation of fabric smoothing with three difficulty tiers of initial fabric state complexity, (2) deep imitation learning of fabric smoothing policies from an algorithmic supervisor using a sequence of pick and pull actions, and (3) transfer to physical experiments on a da Vinci surgical robot with comparisons of coverage performance using color (RGB), depth (D), or RGBD input images. Supplementary material is available on the project website `https://sites.google.com/view/fabric-smoothing`.

3.1 Related Work

Well-known research on robotic fabric manipulation [21, 177, 29] uses bilateral robots and gravity to expose corners. Osawa et al. [155] proposed a method of iteratively re-grasping the

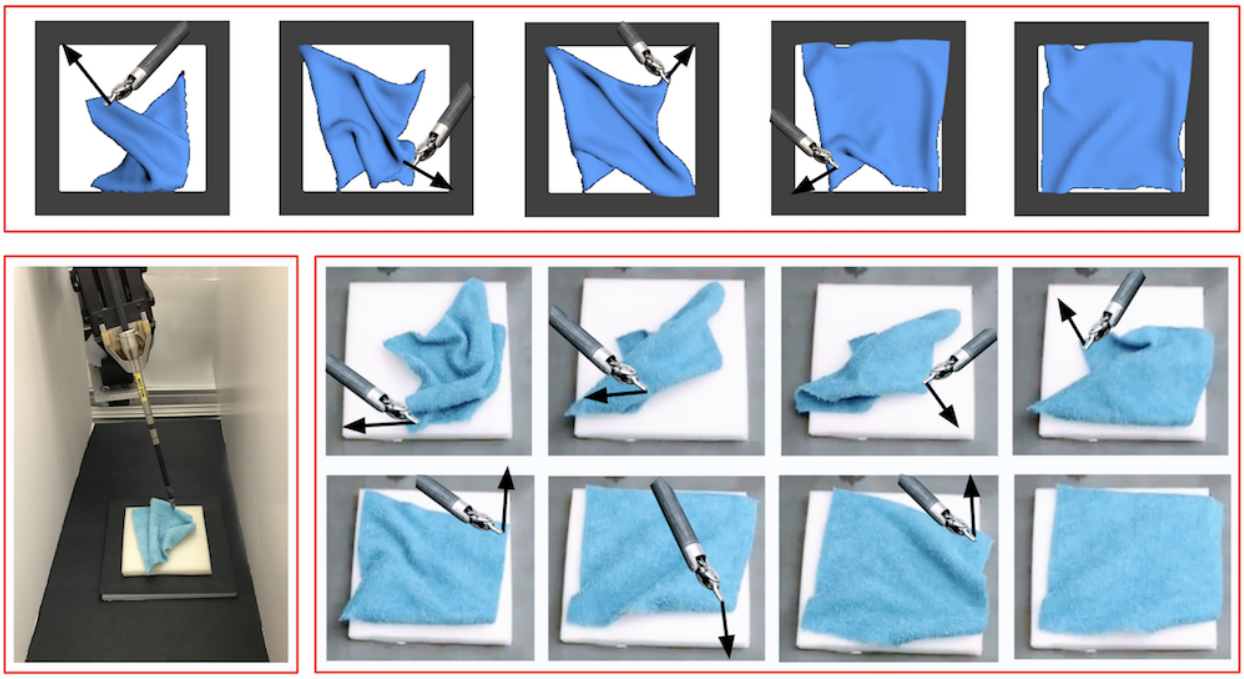

Figure 3.1: Learned policies executed in simulation and with a physical da Vinci surgical robot, with actions indicated from the overlaid arrows. Policies are learned in simulation using DAgger with an algorithmic supervisor that has full state information, using structured domain randomization with color and/or depth images. The 4-action smoothing episode in simulation (top) increases coverage from 43% to 95%. The 7-action episode on the physical da Vinci robot (bottom) increases coverage from 49% to 92%.

lowest hanging point of a fabric to flatten and classify fabrics. Subsequently, Kita et al. [104, 105] used a deformable object model to simulate fabric suspended in the air, allowing the second gripper to grasp at a desired point. Follow-up work generalized to a wider variety of initial configurations of new fabrics. In particular, Maitin-Shepard et al. [136], Cusumano-Towner et al. [31], and Doumanoglou et al. [37] identified and tensioned corners to fold laundry or to bring clothing to desired positions. These methods rely on gravity to reveal corners of the fabric. We consider the setting where a single armed robot adjusts a fabric strewn across a surface without lifting it entirely in midair, which is better suited for larger fabrics or when robots have a limited range of motion.

3.1.1 Reinforcement Learning for Fabric Manipulation

Reinforcement Learning (RL) [201] is a promising method for training policies that can manipulate highly deformable objects. In RL applications for folding, Matas et al. [139] assumed that fabric is flat, and Balaguer et al. [9] began with fabric gripped in midair

to loosen wrinkles. In contrast, we consider the problem of bringing fabric from a highly rumpled configuration to a flat configuration. Using model-based RL, Ebert et al. [38] were able to train robots to fold pants and fabric. This approach requires executing a physical robot for many thousands of actions and then training a video prediction model. In surgical robotics, Thananjeyan et al. [205] used RL to learn a tensioning policy to cut gauze, with one arm pinching at a pick point to let the other arm cut. We focus on fabric smoothing without tensioning, and additionally consider cases where the initial fabric state may be highly rumpled and disordered. In concurrent and independent work, Jangir et al. [93] used deep reinforcement learning with demonstrations to train a policy using fabric state information for simulated dynamic folding tasks.

3.1.2 Fabric Smoothing

In among the most relevant prior research on fabric smoothing, Willimon et al. [220] present an algorithm that pulls at eight fixed angles, and then uses a six-step stage to identify corners from depth images using the Harris Corner Detector [71]. They present experiments on three simulated trials and one physical robot trial. Sun et al. [197] followed up by attempting to explicitly detect and then pull at wrinkles. They measure wrinkledness as the average absolute deviation in a local pixel region for each point in a depth map of the fabric [167] and apply a force perpendicular to the largest wrinkle. Sun et al. evaluate on eight fixed, near-flat fabric starting configurations in simulation. In subsequent work, Sun et al. [198] improved the detection of wrinkles by using a shape classifier as proposed in Koenderink and van Doorn [108]. Each point in the depth map is classified as one of nine shapes, and they use contiguous segments of certain shapes to define a wrinkle. While Sun et al. were able to generalize the method beyond a set of hard-coded starting states, it was only tested on nearly flat fabrics in contrast to the highly rumpled configurations we explore.

In concurrent and independent work, Wu et al. [227] trained an image-based policy for fabric smoothing in simulation using deep reinforcement learning, and then applied domain randomization to transfer it to a physical PR2 robot.

This chapter extends prior work by Seita et al. [185] that only estimated a pick point and pre-defined the pull vector. In contrast, we learn the pull vector and pick point simultaneously. Second, by developing a simulator, we generate far more training data and do not need to run a physical robot. This enables us to perform systematic experiments comparing RGB, D, and RGBD image inputs.

3.2 Problem Statement

Given a deformable fabric and a flat fabric plane, each with the same rectangular dimensions, we consider the task of manipulating the fabric from a start state to one that maximally covers the fabric plane. We define an *episode* as one instance of the fabric smoothing task.

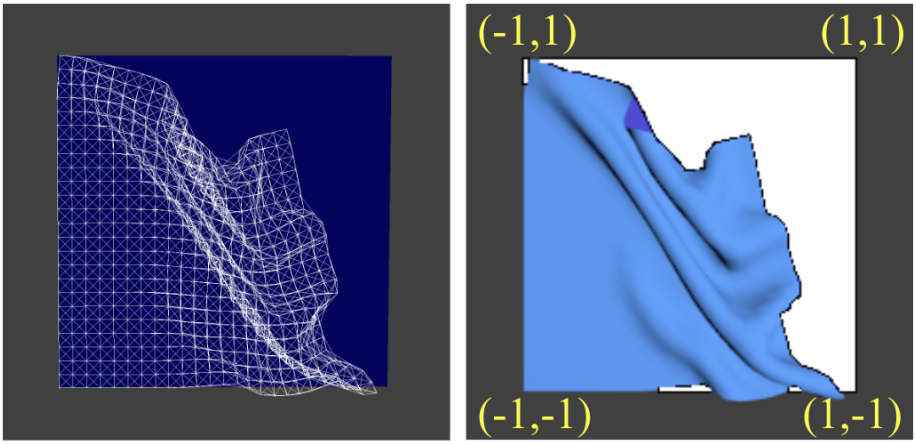

Figure 3.2: FEM fabric simulation. Left: a wireframe rendering, showing the 25×25 grid of points and the spring-mass constraints. Right: the corresponding image with the white fabric plane, rendered using Blender and overlaid with coordinates. The coverage is 73%, measured as the percentage of the fabric plane covered.

Concretely, let ξ_t be the full state of the fabric at time t with positions of all its points (described in Section 3.3). Let $\mathbf{o}_t \in O$ be the *image observation* of the fabric at time t, where $O = \mathbb{R}^{H \times W \times c}$ represents the space of images with $H \times W$ pixels, and $c = 1$ channels for depth images, $c = 3$ for color images, or $c = 4$ for combined color and depth (i.e., RGBD) images. Let A be the set of actions the robot may take (see Section 3.3.1). The task performance is measured with *coverage* $C(\xi_t)$, or the percentage of the fabric plane covered by ξ_t.

We frame this as imitation learning [6], where a supervisor provides data in the form of paired observations and actions $\mathcal{D} = \{(\mathbf{o}_t, \mathbf{a}_t)\}_{t=1}^{N}$. From $\mathcal{D}$, the robot's goal is to learn a policy $\pi : O \rightarrow A$ that maps an observation to an action, and executes sequentially until a coverage threshold or iteration termination threshold is reached.

3.3 Fabric and Robot Simulator

We implement a Finite Element Method (FEM) [12] fabric simulator and interface with an OpenAI gym environment design [22]. The simulator is open source and available on the project website. Alternative fabric simulators exist, such as from Blender [28], which is a popular open-source computer graphics software toolkit. Since 2017, Blender has had significant improvements in physics and realism of fabrics, but these changes are only supported in Blender 2.80, which does not support headless rendering of images and therefore meant we could not run massive data collection. We downgraded to an older version of Blender, 2.79, which supports headless rendering, to create images generated from the proposed custom-built fabric simulator. MuJoCo 2.0 provides another fabric simulator, but did not support

OpenAI-gym style fabric manipulation environments until concurrent work [227].

The fabric (Figure 3.2) is represented as a grid of 25×25 point masses, connected by three types of springs [165]:

- *Structural*: between a point mass and the point masses to its left and above it.
- *Shear*: between a point mass and the point masses to its diagonal upper left and diagonal upper right.
- *Flexion*: between a point mass and the point masses two away to its left and two above it.

The springs modeling the bending constraints are weaker than their structural and shearing counterparts.

Each point mass is acted upon by both an external gravitational force which is calculated using Newton's Second Law and a spring correction force

$$F_s = k_s \cdot (\|q_a - q_b\|_2 - \ell), \tag{3.1}$$

for each of the springs representing the constraints above, where k_s is a spring constant, $q_a \in \mathbb{R}^3$ and $q_b \in \mathbb{R}^3$ are positions of any two point masses connected by a spring, and ℓ is the default spring length. We update the point mass positions using Verlet integration [214]. Verlet integration computes a point mass's new position at time $t + \Delta_t$, denoted with $p_{t+\Delta_t}$, as:

$$p_{t+\Delta_t} = p_t + v_t\Delta_t + a_t\Delta_t^2, \tag{3.2}$$

where $p_t \in \mathbb{R}^3$ is the position, $v_t \in \mathbb{R}^3$ is the velocity, $a_t \in \mathbb{R}^3$ is the acceleration from all forces, and $\Delta_t \in \mathbb{R}$ is a timestep. Verlet integration approximates $v_t\Delta_t = p_t - p_{t-\Delta_t}$ where $p_{t-\Delta_t}$ is the position at the last time step, resulting in

$$p_{t+\Delta_t} = 2p_t - p_{t-\Delta_t} + a_t\Delta_t^2 \tag{3.3}$$

The simulator adds damping to simulate loss of energy due to friction, and scales down v_t, leading to the final update:

$$p_{t+\Delta_t} = p_t + (1-d)(p_t - p_{t-\Delta_t}) + a_t\Delta_t^2 \tag{3.4}$$

where $d \in [0, 1]$ is a damping term, which we tuned to 0.02 based on visually inspecting the simulator.

We apply a constraint from Provot [165] by correcting point mass positions so that spring lengths are at most 10% greater than ℓ at any time. We also implement fabric-fabric collisions following [11] by adding a force to "separate" two points if they are too close.

The simulator provides access to the full fabric state ξ_t, which contains the exact positions of all 25×25 points, but does not provide image observations $\mathbf{o}_t$ which are more natural and realistic for transfer to physical robots. To obtain image observations of a given fabric state, we create a triangular mesh and render using Blender.

3.3.1 Actions

We define an action at time t as a 4D vector which includes the pick point (x_t, y_t) represented as the coordinate over the fabric plane to grasp, along with the pull direction. The simulator implements actions by grasping the top layer of the fabric at the pick point. If there is no fabric at (x_t, y_t), the grasp misses the fabric. After grasping, the simulator pulls the picked point upwards and towards direction $\Delta x_t \in [-1, 1]$ and $\Delta y_t \in [-1, 1]$, deltas in the x and y direction of the fabric plane.[1] In summary, actions $\mathbf{a}_t \in A$ are defined as:

$$\mathbf{a}_t = \langle x_t, y_t, \Delta x_t, \Delta y_t \rangle \tag{3.5}$$

representing the pick point coordinates (x_t, y_t) and the pull vector $(\Delta x_t, \Delta y_t)$ relative to the the pick point.

3.3.2 Starting State Distributions

The performance of a smoothing policy, or more generally any fabric manipulation policy, depends heavily on the distribution of starting fabric states. We categorize episodes as belonging to one of three custom difficulty tiers. For each tier, we randomize the starting state of each episode. The starting states, as well as their average coverage based on 2000 simulations, are generated as follows:

- **Tier 1,** $78.3 \pm 6.9\%$ **Coverage (High)**: starting from a flat fabric, we make two short, random pulls to slightly perturb the fabric. All fabric corners remain visible.
- **Tier 2,** $57.6 \pm 6.1\%$ **Coverage (Medium)**: we let the fabric drop from midair on one side of the fabric plane, perform one random grasp and pull across the plane, and then do a second grasp and pull to cover one of the two fabric corners furthest from its plane target.
- **Tier 3,** $41.1 \pm 3.4\%$ **Coverage (Low)**: starting from a flat fabric, we grip at a random pick point and pull high in the air, drag in a random direction, and then drop, usually resulting in one or two corners hidden.

Figure 3.3 shows examples of color and depth images of fabric initial states in simulation and real physical settings for all three tiers of difficulty. The supplementary material contains additional examples of images.

3.4 Baseline Policies

We propose five baseline policies for fabric smoothing. All baselines, except the random one, must use the fabric state ξ_t in simulation.

[1] We tried an alternative parameterization based on length and angle, but the angle is problematic at the discontinuity of $-\pi$ and π.

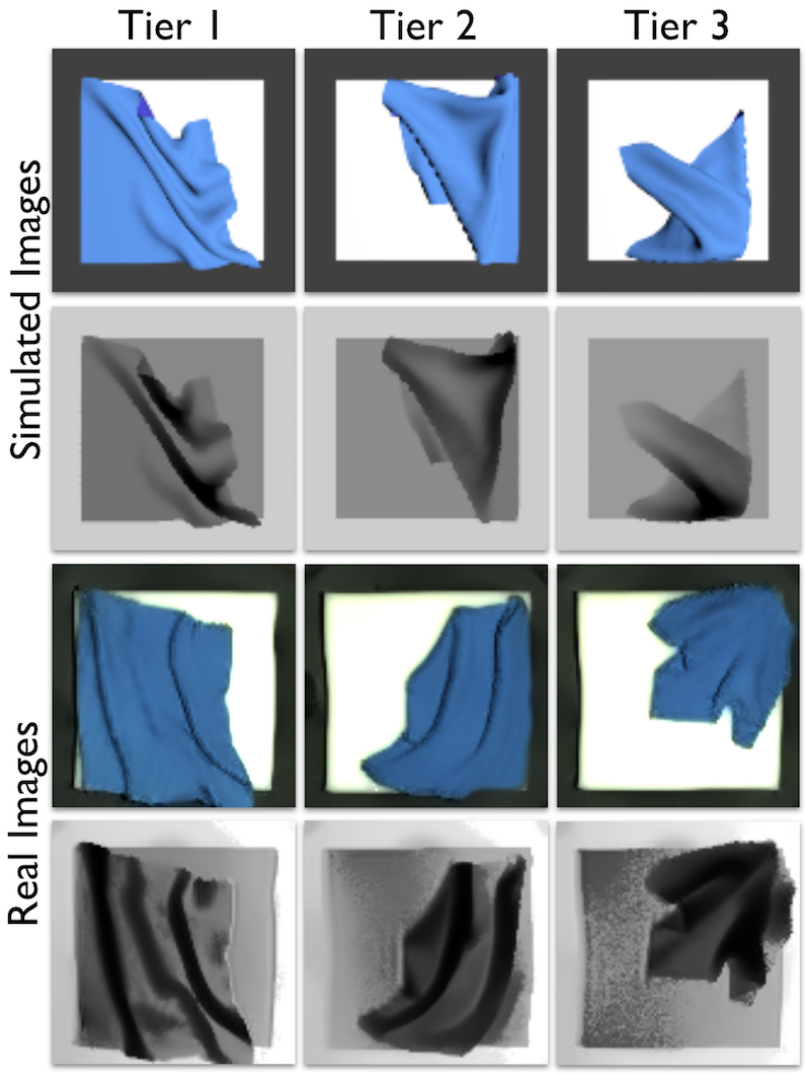

Figure 3.3: Initial fabric configurations drawn from the distributions specified in Section 3.3.2, with tiers grouped by columns. The first two rows show representative simulated color (RGB) and depth (D) images, respectively, while the last two rows show examples of real images from a mounted Zivid One Plus camera, after smoothing and de-noising, which we then pass as input to a neural network policy. Domain randomization is not applied on the simulated images shown here.

Random

As a naive baseline, we test a random policy that uniformly selects random pick points and pull directions.

Highest (Max z)

This policy, tested in Seita et al. [185] grasps the highest point on the fabric. We get the pick point by determining p, the highest of the $25^2 = 625$ points from ξ_t. To compute the pull vector, we obtain the target coordinates by considering where p's coordinates would be if the fabric is perfectly flat. The pull vector is then the vector from p's current position to that target. Seita et al. [185] showed that this policy can achieve reasonably high coverage, particularly when the highest point corresponds to a corner fold on the uppermost layer of the fabric.

Wrinkle

One baseline method is to detect wrinkles and pull in the direction perpendicular to the largest wrinkle. Intuitively, this flattens the wrinkle, and the process can be repeated for subsequent (perhaps smaller) wrinkles. Sun et al. [197] propose a two-stage algorithm to first identify wrinkles and then to derive a force parallel to the fabric plane to flatten the largest wrinkle. The process repeats for subsequent wrinkles. We implement this method by finding the point in the fabric of largest local height variance. Then, we find the neighboring point with the next largest height variance, treat the vector between the two points as the wrinkle, and pull perpendicular to it.

Oracle

This policy uses complete state information from ξ_t to find the fabric corner furthest from its fabric plane target, and pulls it towards that target. When a corner is occluded and underneath a fabric layer, this policy will grasp the point directly above it on the uppermost fabric layer, and the resulting pull usually decreases coverage.

Oracle-Expose

When a fabric corner is occluded, and other fabric corners are not at their targets, this policy picks above the hidden corner, but pulls away from the fabric plane target to reveal the corner for a subsequent action.

3.5 Simulation Results for Baseline Policies

We evaluate the five baseline fabric smoothing policies by running each for 2000 episodes in simulation. Each episode draws a randomized fabric starting state from one of three difficulty tiers (Section 3.3.2), and lasts for a maximum of 10 actions. Episodes can terminate earlier under two conditions: (1) if a pre-defined coverage threshold is obtained, or (2) the fabric is out of bounds over a certain threshold. For (1) we use 92% as the threshold, which produces visually smooth fabric (e.g., see the last few images in Figure 3.4) and avoids supervisor data

Table 3.1: Results from the five baseline policies discussed in Section 3.4. We report final coverage and the number of actions per episode. All statistics are from 2000 episodes, with tier-specific starting states. Both oracle policies (in bold) perform the best.

Tier	Method	Coverage	Actions
1	Random	25.0 +/- 14.6	2.43 +/- 2.2
1	Highest	66.2 +/- 25.1	8.21 +/- 3.2
1	Wrinkle	91.3 +/- 7.1	5.40 +/- 3.7
1	Oracle	**95.7 +/- 2.1**	**1.76 +/- 0.8**
1	Oracle-Expose	**95.7 +/- 2.2**	**1.77 +/- 0.8**
2	Random	22.3 +/- 12.7	3.00 +/- 2.5
2	Highest	57.3 +/- 13.0	9.97 +/- 0.3
2	Wrinkle	87.0 +/- 10.8	7.64 +/- 2.8
2	Oracle	**94.5 +/- 5.4**	**4.01 +/- 2.0**
2	Oracle-Expose	**94.6 +/- 5.0**	**4.07 +/- 2.2**
3	Random	20.6 +/- 12.3	3.78 +/- 2.8
3	Highest	36.3 +/- 16.3	7.89 +/- 3.2
3	Wrinkle	73.6 +/- 19.0	8.94 +/- 2.0
3	Oracle	**95.1 +/- 2.3**	**4.63 +/- 1.1**
3	Oracle-Expose	**95.1 +/- 2.2**	**4.70 +/- 1.1**

being dominated by taking actions of short magnitudes at the end of its episodes. For (2) we define a fabric as out of bounds if it has any point which lies at least 25% beyond the fabric plane relative to the full distance of the edge of the plane. This threshold allows the fabric to go slightly off the fabric plane which is sometimes unavoidable since a perfectly smoothed fabric is the same size as the plane. We do not allow a pick point to lie outside the plane.

Table 3.1 indicates that both oracle policies attain nearly identical performance and have the highest coverage among the baseline policies, with about 95% across all tiers. The wrinkles policy is the next best policy in simulation, with 91.3%, 87.0%, and 73.6% final coverage for the three respective tiers, but requires substantially more actions per episode.

One reason why the oracle policy still performs well with occluded corners is that the resulting pulls can move those corners closer to their fabric plane targets, making it easier for subsequent actions to increase coverage. Figure 3.4 shows an example episode from the oracle policy on a tier 3 starting state. The second action pulls at the top layer of the fabric above the corner, but the resulting action still moves the occluded corner closer to its target. This allows subsequent actions to adjust the fabric to attain high coverage.

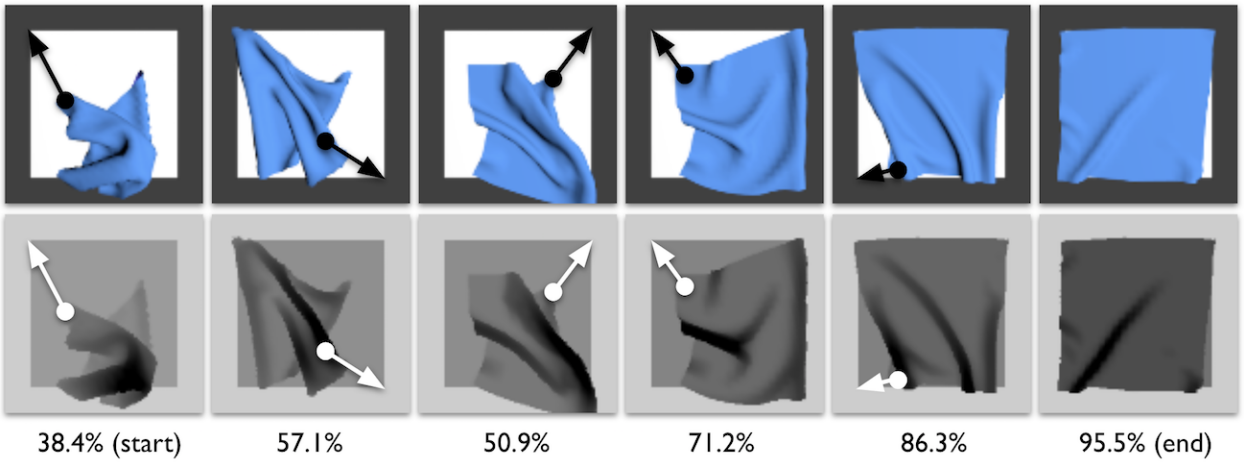

Figure 3.4: Example simulated episode of the oracle corner supervisor policy, from left to right, drawn from a Tier 3 starting state with 38.4% coverage. The policy uses the exact corner location from the fabric state (not the images) and pulls the one furthest from its target on the fabric plane. For visualization purposes, we show matching color and depth images from the episode without domain randomization. Overlaid circles and arrows represent the action taken after the given state. In the second action, the fabric corner furthest from the target is slightly underneath the fabric, and the supervisor pulls at the fabric's top layer. Nonetheless, the underlying corner is still moved closer to its target position, and subsequent pulls are able to achieve high coverage. The oracle policy took five actions before getting to 95.5% coverage and triggering the 92% threshold in the rightmost images.

3.6 Imitation Learning with DAgger

We use the oracle (not oracle-expose) policy to generate supervisor data and corrective labels. For each tier, we generate 2000 episodes from the supervisor and use that as offline data. We train a fabric smoothing policy in simulation using imitation learning on synthetic images. When behavior cloning on supervisor data, the robot's policy will learn the supervisor's actions on states in the training data, but generalize poorly outside the data distribution [162]. To address this, we use Dataset Aggregation (DAgger) [170], which requests the supervisor to label the states the robot encounters when running its learned policy. A limitation of DAgger is the need for continued access to the supervisor's policy, rather than just offline data. During training, the oracle corner-pulling supervisor is able to efficiently provide an action to each data point encountered by accessing the underlying fabric state information, so in practice this does not cause problems.

3.6.1 Policy Training Procedure

We use domain randomization [207] during training. For RGB images, we randomize the fabric color by selecting RGB values uniformly at random across intervals that include shades of blue, purple, pink, red, and gray. For RGB images, we additionally randomize the brightness with gamma corrections [164] and vary the shading of the fabric plane. For depth (D) images, we make the images slightly darker to more closely match real depth images. For both RGB and D, we randomize the camera pose with independent Gaussian distributions for each of the position and orientation components. The supplementary material has examples of images after domain randomization.

The policy neural network architecture is similar to the one in Matas et al. [139]. As input, the network consumes images with dimension $(100 \times 100 \times c)$, where the number of channels is $c = 1$ for D, $c = 3$ for RGB, or $c = 4$ for RGBD images. It passes the input image through four convolutional layers, each with 32 filters of size 3×3, followed by four dense layers of size 256 each, for a total of about 3.4 million parameters. The network produces a 4D vector with a hyperbolic tangent applied to make components within $[-1, 1]$. We optimize using Adam with learning rate 10^{-4} and use L_2 regularization of 10^{-5}.

The imitation learning code uses OpenAI baselines [35] to make use of its parallel environment support. We run the fabric simulator in ten parallel environments, which helps to alleviate the major time bottleneck when training, and pool together samples in a shared dataset. We first train with a "behavior cloning (BC) phase" where we minimize the L_2 error on the offline supervisor data, and then use a "DAgger phase" which rolls out the agent's policy and applies DAgger. We use 500 epochs of behavior cloning based on when the network's L_2 error roughly converged on a held-out validation dataset. The DAgger phase runs until the agent collectively performs 50,000 total steps. Further training details are in the supplementary material.

3.6.2 Simulation Experiments

For all simulated training runs, we evaluate on 50 new tier-specific starting states that are not seen during training. Figure 3.5 shows results across all tiers, suggesting that after behavior cloning, DAgger improves final coverage performance by 6.1% when averaging over the nine experimental conditions (three image input modalities across three tiers). In addition, RGB policies attain better coverage in simulation than D policies with gains of 10.8%, 8.3%, and 10.9% across respective tiers, which may be due to high color contrast between the fabric and fabric plane in the RGB images, as opposed to the D images (see Figure 3.3). The RGBD policies perform at a similar level as the RGB-based policies.

In all difficulty tiers, the RGB policies get higher final coverage performance than the wrinkles policy (from Table 3.1): 94.8% over 91.3%, 89.6% over 87.0%, and 91.2% over 73.6%, respectively, and gets close to the corner pulling supervisor despite only having access to image observations rather than underlying fabric state. Similarly, the RGBD policies

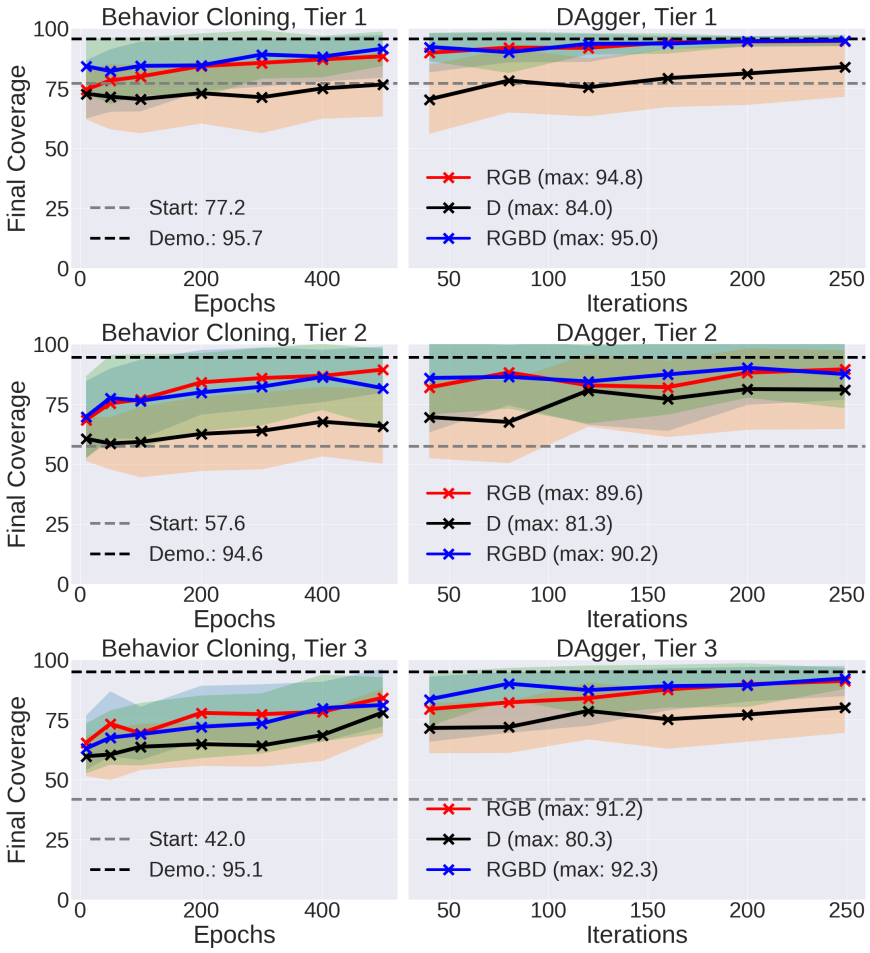

Figure 3.5: Coverage over 50 simulated episodes at checkpoints (shown with "X") during behavior cloning (left) and DAgger (right), which begins right after the last behavior cloning epoch. Results, from top to bottom, are for tier 1, 2, and 3 starting states. We additionally annotate with dashed lines the average starting coverage and the supervisor's average final coverage. Results suggest that the RGB and RGBD policies attain strong coverage performance in simulation. Shaded regions represent one standard deviation range.

outperform the wrinkles policy across all tiers. The depth-only policies outperform the wrinkles policy on tier 3, with 80.3% versus 73.6% coverage.

3.7 Physical Experiments

The da Vinci Research Kit (dVRK) surgical robot [97] is a cable-driven surgical robot with imprecision as reviewed in prior work [134, 186]. We use a single arm with an end effector

that can be opened to 75 or a gripper width of 10mm. We set a fabric plane at a height and location that allows the end-effector to reach all points on it. To prevent potential damage to the grippers, the fabric plane is foam rubber, which allows us to liberally set the gripper height to be lower and avoids a source of height error present in [139]. For the fabric, we cut a 5x5 inch piece from a Zwipes 735 Microfiber Towel Cleaning Cloth with a blue color within the distribution of domain randomized fabric colors. We mount a Zivid One Plus RGBD camera 0.9 meters above the workspace, which is used to obtain color and depth images.

3.7.1 Physical Experiment Protocol

We manually create starting fabric states similar to those in simulation for all tiers. Given a starting fabric, we randomly run one of the RGB or D policies for one episode for at most 10 steps (as in simulation). Then, to make comparisons fair, we "reset" the fabric to be close to its starting state, and run the other policy. After these results, we then run the RGBD baseline to combine RGB and D images, again manipulating the fabric starting state to be similar among comparable episodes. To facilitate this process, we save all real images encountered and use them as a guide to creating the initial fabric configurations.

During preliminary trials, the dVRK gripper would sometimes miss the fabric by 1-2 mm, which is within the calibration error. To counter this, we measure structural similarity [218] of the image before and after an action to check if the robot moved the fabric. If it did not, the next action is adjusted to be closer to the center of the fabric plane, and the process repeats until the robot touches fabric.

3.7.2 Physical Experiment Results

We run 20 episodes for each combination of input modality (RGB, D, or RGBD) and tiers, resulting in 180 total episodes as presented in Table 3.2. We report starting coverage, ending coverage, maximum coverage across the episode after the initial state, and the number of actions. The maximum coverage allows for a more nuanced understanding of performance, because policies can take strong initial actions that achieve high coverage (e.g., above 80%) but a single counter-productive action at the end can substantially lower coverage.

Results suggest that, despite not being trained on real images, the learned policies can smooth physical fabric. All policies improve over the starting coverage across all tiers. Final coverage averaged across all tiers is 86.3%, 69.0%, and 89.8% for RGB, D, and RGBD policies, respectively, with net coverage gains of 25.2%, 7.8%, and 33.4% over starting coverage. In addition, the RGB and RGBD policies deployed on tier 1 starting states each achieve the 92% coverage threshold 20 out of 20 times. While RGB has higher final coverage on tier 1 starting states, the RGBD policies appear to have stronger performance on the more difficult starting states without taking considerably more actions.

Qualitatively, the RGB and RGBD-trained policies are effective at "fine-tuning" by taking several short pulls to trigger at least 92% coverage. For example, Figure 3.6 shows an episode taken by the RGBD policy trained on tier 3 starting states. It is able to smooth the highly

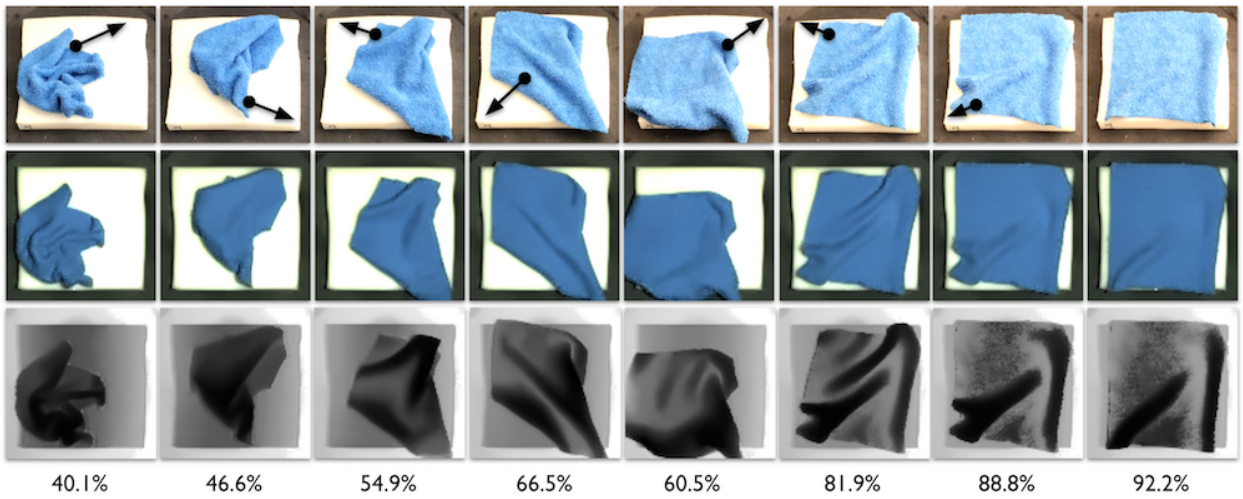

Figure 3.6: An example episode taken by a learned policy trained on RGBD images from tier 3 starting states. The top row shows screen captures from the camera view used to record videos. The middle and bottom row show the processed RGB and D images that are passed into the neural network policy. The leftmost images show the starting state of the fabric, set to be highly wrinkled with at least the bottom left fabric corner hidden. The policy takes seven actions shown here, with pick points and pull vectors indicated by the overlaid black arrows in the top row. Despite the highly wrinkled starting state, along with hidden fabric corners (as shown in the first four columns) the policy is able to smooth fabric from 40.1% to 92.2% coverage as shown at the rightmost images.

wrinkled fabric despite several corners hidden underneath fabric layers. The depth-only policies do not perform as well, but this is in large part because the depth policy sometimes takes counterproductive actions after several reasonable actions. This may be in part due to uneven texture on the fabric we use, which is difficult to replicate in simulated depth images.

The supplementary material contains videos of every tier and policy combination.

3.7.3 Failure Cases

The policies are sometimes susceptible to performing highly counter-productive actions. In particular, the depth-only policies can fail by pulling near the center of the fabric for fabrics that are already nearly smooth, as shown in Figure 3.7. This results in poor coverage and may lead to cascading errors where one poor action can lead fabric to reach states that are not seen in training.

One cause may be that there are several fabric corners that are equally far from their targets, which creates ambiguity in which corner should be pulled. One approach to mitigate this issue, in addition to augmenting depth with RGB-based data as in the RGBD policies we train, is to formulate corner picking with a mixture model to resolve this ambiguity.

Table 3.2: Physical experiments. We run 20 episodes for each of the tier 1 (T1), tier 2 (T2), and tier 3 (T3) fabric conditions, with RGB, D, and RGBD policies, for a total of $20 \times 3 \times 3 = 180$ episodes. We report: (1) starting coverage, (2) final coverage, with the highest values in each tier in bold, (3) maximum coverage at any point after the start state, and (4) the number of actions per episode. Results suggest that RGBD is the best policy with harder starting fabric configurations.

Setting	**(1) Start**	**(2) Final**	**(3) Max**	**(4) Actions**
T1 RGB	78.4 +/- 4	**96.2 +/- 2**	96.2 +/- 2	1.8 +/- 1
T1 D	77.9 +/- 4	78.8 +/- 24	90.0 +/- 10	5.5 +/- 4
T1 RGBD	72.5 +/- 4	95.0 +/- 2	95.0 +/- 2	2.1 +/- 1
T2 RGB	58.5 +/- 6	87.7 +/- 13	92.7 +/- 4	6.3 +/- 3
T2 D	58.7 +/- 5	64.9 +/- 20	85.7 +/- 8	8.3 +/- 3
T2 RGBD	55.0 +/- 5	**91.3 +/- 8**	92.7 +/- 6	6.8 +/- 3
T3 RGB	46.2 +/- 4	75.0 +/- 18	79.9 +/- 14	8.7 +/- 2
T3 D	47.0 +/- 3	63.2 +/- 9	74.7 +/- 10	10.0 +/- 0
T3 RGBD	41.7 +/- 2	**83.0 +/- 10**	85.8 +/- 6	8.8 +/- 2

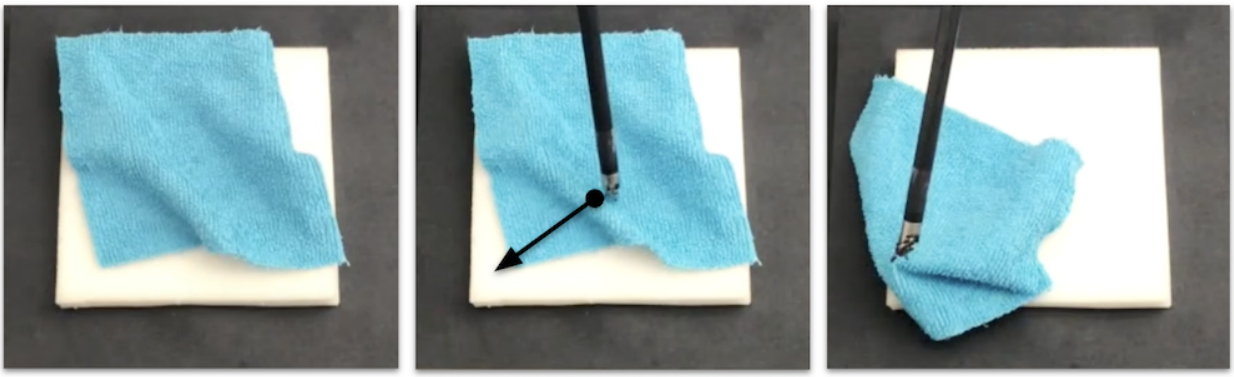

Figure 3.7: A poor action from a policy trained on only depth images. Given the situation shown in the left image, the learned policy erroneously picks a point near the center of the fabric. The resulting pick, followed by the pull to the lower left, causes a major decrease in coverage and makes it hard for the robot to recover.

3.8 Conclusion and Future Work

In this chapter, we investigate baseline and learned policies for fabric smoothing. Using a low fidelity fabric simulator and a custom environment, we train policies in simulation using DAgger with a corner pulling supervisor. We use domain randomization to transfer policies to a surgical robot. When testing on fabric of similar color to those used in training, RGB-based and RGBD-based policies achieve higher coverage than D-based policies. On

the harder starting fabric configurations, the combined RGBD-based policies get the highest coverage among the tested policies, suggesting that depth is beneficial.

In future work, we will test on fabric shapes where corner pulling policies may get poor coverage. We plan to apply state-of-the-art deep reinforcement learning methods to learn richer policies that can explicitly reason over multiple time steps and varying geometries. To improve sample efficiency, we will consider model-based approaches such as Visual Foresight [38] and future image similarity [222], along with approaches that predict physics properties [88]. We will utilize higher-fidelity fabric simulators such as ARCSim [150]. Finally, we would like to extend the method beyond fabric coverage to tasks such as folding and wrapping, and will apply it to ropes, strings, and other deformable objects.

Chapter 4

VisuoSpatial Foresight for Multi-Step, Multi-Task Fabric Manipulation

Advances in robotic manipulation of deformable objects has lagged behind work on rigid objects due to the far more complex dynamics and configuration space. Fabric manipulation in particular has applications ranging from senior care [63], sewing [182], ironing [124], bed-making [185] and laundry folding [143, 123, 230, 191] to manufacturing upholstery [209] and handling surgical gauze [205]. However, prior work in fabric manipulation has generally focused on designing policies that are only applicable to a *specific* task via manual design [143, 123, 230, 191] or policy learning [184, 227].

The difficulty in developing accurate analytical models of highly deformable objects such as fabric motivates using data-driven strategies to estimate models, which can then be used for general purpose planning. While there has been prior work in system identification for robotic manipulation [107, 82, 17, 169, 26, 27], many of these techniques rely on reliable state estimation from observations, which is especially challenging for deformable objects. One popular alternative is visual foresight [38, 49], which uses a large dataset of self-supervised interaction data in the environment to learn a visual dynamics model directly from raw image observations and has shown the ability to generalize to a wide variety of environmental conditions [33]. This model can then be used for planning to perform different tasks at test time. The technique has been successfully applied to learning the dynamics of complex tasks, such as pushing and basic fabric folding [38, 49]. However, two limitations of prior work in visual foresight are: 1) the data requirement for learning accurate visual dynamics models is often very high, requiring several days of continuous data collection on real robots [33, 38], and 2) while prior work reports experiments on basic fabric manipulation tasks [38], these are relatively short-horizon tasks with a wide range of valid goal images in contrast to the multi-step tasks we consider to achieve specific goal images.

In this chapter, we present a system (Figure 4.1) that takes steps towards addressing these challenges by integrating RGB and depth sensing to learn visual-spatial (visuospatial) dynamics models in simulation using only random interaction data for training and domain randomization techniques for sim-to-real transfer. This chapter introduces: 1) "VisuoSpatial

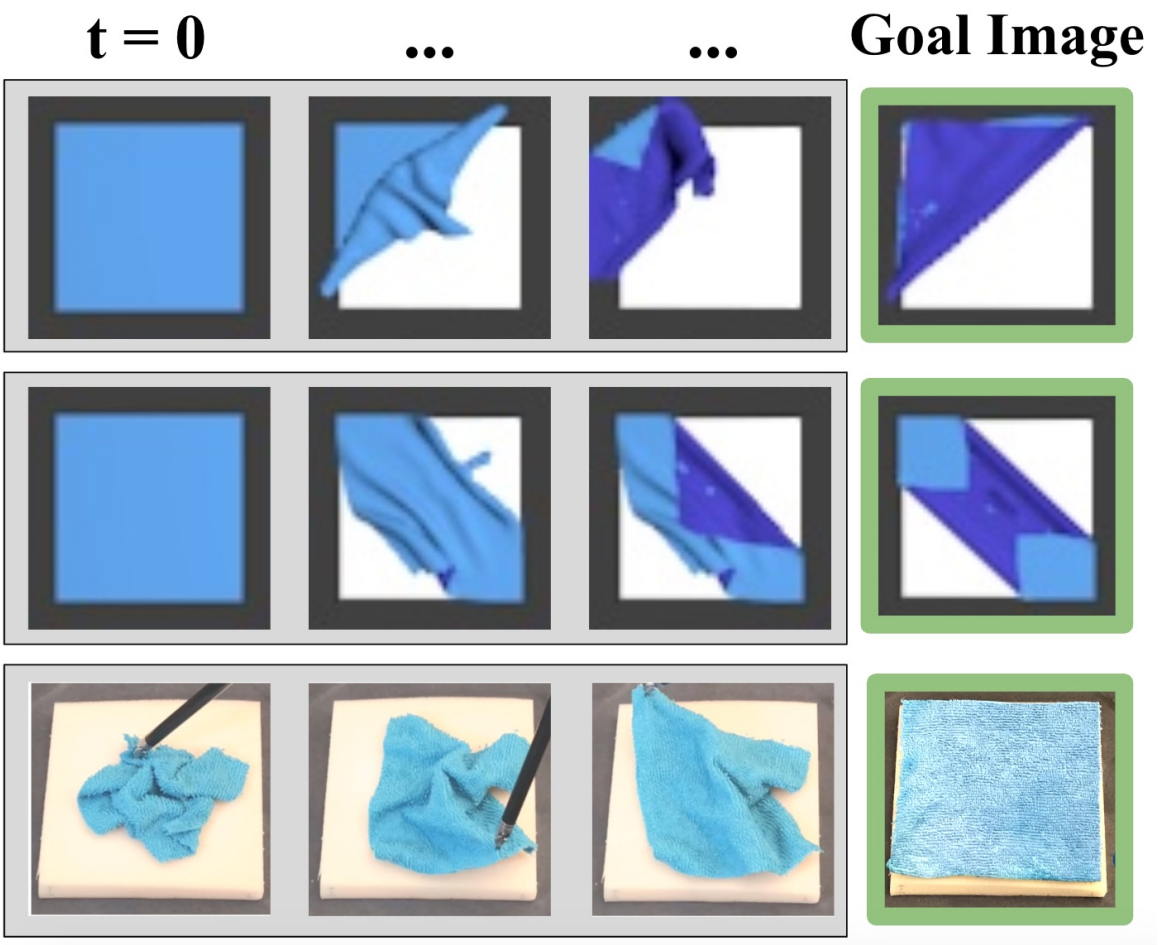

Figure 4.1: Using VSF on domain randomized RGBD data, we learn a goal-conditioned fabric manipulation policy in simulation without any task demonstrations. We evaluate the same policy on several goal images in simulation (top two rows) and on the physical da Vinci surgical robot (bottom row). The rows display the RGB portions of subsampled frames from episodes for folding, double folding, and smoothing, respectively, toward the goal images in the right column.

Foresight" on domain randomized, simulated RGB and depth data to facilitate rapid data collection and transfer to physical systems, 2) simulated experiments demonstrating how the learned dynamics can be used to define a goal-conditioned fabric manipulation policy which can perform a variety of multi-step fabric smoothing and folding tasks, and 3) results suggesting that the learned policy transfers to a real physical system with promising smoothing and folding results. In physical smoothing experiments, the learned policy is able to outperform an imitation learning baseline by 2% despite using no task demonstrations.

Code, data, videos, and supplementary material are available on the project website at `https://sites.google.com/view/fabric-vsf/`.

4.1 Related Work

Manipulating fabric is a long-standing challenge in robotics. Fabric smoothing in particular has received attention since it helps standardize the configuration of the fabric for subsequent tasks such as folding [21, 177]. A popular approach in prior work is to first hang fabric in the air and allow gravity to "vertically smooth" it [155, 104, 105, 37]. Maitin-Shepard et al. [136] use this approach to achieve a 100% success rate in single-towel folding. In contrast, the approach we present is targeted towards larger fabrics like blankets [185] and for single-armed robots which may have a limited range of motion, making such "vertical smoothing" infeasible. Similar work on fabric smoothing and folding, such as by [9] and Jia et al. [94, 95] assume the robot is initialized with the fabric already grasped, while we initialize the robot's end-effector away from the fabric. Willimon et al. [220] and Sun et al. [197, 198] address a similar problem setting as we do, but with initial fabric configurations much closer to fully smoothed than those we consider.

There has been recent interest in learning sequential fabric manipulation policies with fabric simulators. For example, Seita et al. [184] and Wu et al. [227] learn fabric smoothing in simulation, the former using DAgger [170] and the latter using model-free reinforcement learning (MFRL). Similarly, Matas et al. [139] and Jangir et al. [93] learn policies for folding fabrics using MFRL augmented with task-specific demonstrations. All of these works obtain large training datasets from fabric simulators; examples of simulators with support for fabric include ARCSim [150], MuJoCo [208], PyBullet [30], and Blender [28]. While these algorithms achieve impressive results, they are designed or trained for specific fabric manipulation tasks (such as folding or smoothing), and do not reuse learned structure to generalize to a wide range of tasks. This motivates learning fabric dynamics to enable more general purpose fabric manipulation strategies.

Model-predictive control (MPC) is a popular approach for leveraging learned dynamics for robotics control that has shown success in learning robust closed-loop policies even with substantial dynamical uncertainty [10, 41, 192, 206]. However, many of these prior works require knowledge or estimation of underlying system state, which can often be infeasible or inaccurate. Finn et al. [49, 38] introduce *visual foresight*, and demonstrate that MPC can be used to plan over learned video prediction models to accomplish a variety of robotic tasks, including deformable object manipulation such as folding pants. However, the trajectories shown in [38] are limited to a single pick and pull, while we focus on longer horizon sequential tasks that are enabled by a pick-and-pull action space rather than direct end effector control. Furthermore, the fabric manipulation tasks reported have a wide range of valid goal configurations, such as covering a utensil with a towel or moving a pant leg upwards. In contrast, we focus on achieving precise goal configurations via multi-step interaction with the fabric. Prior work on visual foresight [49, 38, 33] also generally collects data for training visual dynamics models in the real world, which is impractical and unsafe for robots such as the da Vinci surgical robot due to the sheer volume of data required for the technique (on the order of 100,000 to 1 million actions, often requiring several days of physical interaction data [33]). One recent exception is the work of Nair et al. [149] which trains visual dynamics

models in simulation for Tetris block matching. Finally, prior work in visual foresight utilizes visual dynamics model for RGB images, but we find that depth data provides valuable geometric information for fabric manipulation tasks.

4.2 Problem Statement

We consider learning goal-conditioned fabric manipulation policies that enable planning to specific fabric configurations given a goal image of the fabric in the desired configuration. We define the fabric configuration at time t as $\boldsymbol{\xi}_t$, represented via a mass-spring system with an $N \times N$ grid of point masses subject to gravity and Hookean spring forces. Due to the difficulties of state estimation for highly deformable objects such as fabric, we consider overhead RGBD observations $\mathbf{o}_t \in \mathbb{R}^{56\times56\times4}$, which consist of three-channel RGB and single-channel depth images.

We assume tasks have a finite task horizon T and can be achieved with a sequence of actions (from a single robot arm) which involve grasping a specific point on the fabric and pulling it in a particular direction, which holds for common manipulation tasks such as folding and smoothing. We consider four dimensional actions,

$$\mathbf{a}_t = \langle x_t, y_t, \Delta x_t, \Delta y_t \rangle. \tag{4.1}$$

Each action $\mathbf{a}_t$ at time t involves grasping the top layer of the fabric at coordinate (x_t, y_t) with respect to an underlying background plane, lifting, translating by $(\Delta x_t, \Delta y_t)$, and then releasing and letting the fabric settle. If there is no fabric at point (x_t, y_t), then the robot does not grip anything and the translation motion has no effect. When appropriate, we omit the time subscript t for brevity.

The objective is to learn a goal-conditioned policy which minimizes some goal conditioned cost function $c_g(\tau)$ defined on realized interaction episodes with goal g and episode τ, where the latter in this work consists of one or more images.

4.3 Approach

4.3.1 Goal Conditioned Fabric Manipulation

To learn goal-conditioned policies, we build on the visual foresight framework introduced by Finn et al. [49]. Here, a video prediction model (also called a visual dynamics model) is trained, which, given a history of observed images and a sequence of proposed actions, generates a sequence of predicted images that would result from executing the proposed actions in the environment. This model is learned from random interaction data of the robot in the environment. Then, MPC is used to plan over this visual dynamics model with some cost function evaluating the discrepancy between predicted images and a desired goal image. In this work, we: 1) learn a visual dynamics model on RGBD images instead of RGB

images as in prior work, and 2) learn visual dynamics entirely in simulation. We find that these choices improve performance on complex fabric manipulation tasks and accelerate data collection while limiting wear and tear on the physical system.

To represent the dynamics of the fabric, we train a deep recurrent convolutional neural network [65] to predict a sequence of RGBD output frames conditioned on a sequence of RGBD context frames and a sequence of actions. This visuospatial dynamics model is trained on thousands of self-supervised simulated episodes of interaction with the fabric, where an episode consists of a contiguous trajectory of observations and actions. We use Stochastic Variational Video Prediction [7] as discussed in Section 4.3.4. For planning, we utilize a goal-conditioned planning cost function $c_g(\hat{\mathbf{o}}_{t+1:t+H})$ with goal g, which in the proposed work is a target image $\mathbf{o}^{(g)}$. The cost is evaluated over the H-length sequence of predicted images $\hat{\mathbf{o}}_{t+1:t+H}$ sampled from the dynamics model conditioned on the current observation $\mathbf{o}_t$ and some proposed action sequence $\hat{\mathbf{a}}_{t:t+H-1}$. While there are a variety of cost functions that can be used for visual foresight [38], we utilize the Euclidean pixel distance between the final predicted RGBD image at timestep t and the goal image $\mathbf{o}^{(g)}$ across all 4 channels as this works well in practice for the tasks we consider. Precisely, we define the planning cost as follows:

$$c_g(\hat{\mathbf{o}}_{t+1:t+i*}) = \|\mathbf{o}^{(g)} - \hat{\mathbf{o}}_{t+i*}\|_2 \tag{4.2}$$

where

$$i* = \min\{H, T-t\} \tag{4.3}$$

As in prior work [49, 38, 33], we utilize the cross entropy method [171] to plan action sequences to minimize $c_g(\hat{\mathbf{o}}_{t+1:t+H})$ over a receding H-step horizon at each time t. See Figure 4.2 for intuition on the VSF planning phase.

4.3.2 Fabric Simulator

The fabric and robot simulator we use is built on top of the open-source code from [184]. We briefly review the most relevant aspects of the simulator, with emphasis on the changes from the original.

Simulator Design

VisuoSpatial Foresight requires a large amount of training data to predict full-resolution RGBD images. Since getting real data is cumbersome and imprecise, we use a fabric simulator to generate data quickly and efficiently. We use the fabric simulator from [184], which was shown to be sufficiently accurate for learning reasonable fabric smoothing policies with imitation learning.

The fabric is represented as a mass-spring system with a 25×25 grid of point masses [165] with springs connecting it to its neighbors. Verlet integration [214] is used to update point mass positions using finite difference approximations and self-collision is implemented by adding a repulsive force between points that are too close [11]. Damping is also applied to

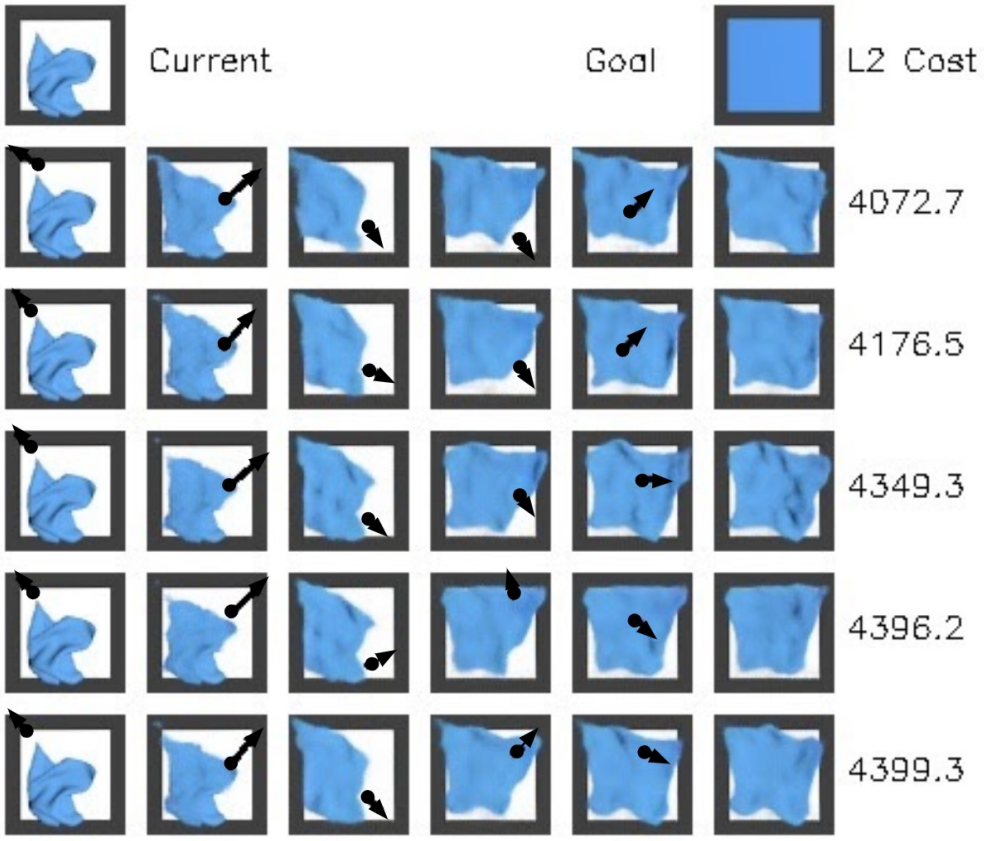

Figure 4.2: Real plans generated by the system at test time for the smoothing task. We generate action sequences with the Cross Entropy Method (CEM) to approximately minimize the cost function, which evaluates L2 distance between the final image in the predicted trajectory and a provided goal image. Here we show the five CEM trajectories with the lowest cost, where the image sequences in each row are outputs of the video prediction model and the black arrows are the pick-and-pull actions projected onto the images.

simulate friction. See Section 4.1 and Appendix C.1 for further detail on alternate fabric simulators and the simulation used in this work.

Manipulation Details

Similar to [184], actions are represented as four-dimensional vectors as described in Section 4.2 with pick point $(x, y) \in [-1, 1]^2$ and pull vector $(\Delta x, \Delta y) \in [-0.4, 0.4]^2$. The only change in the action space is that we truncate Δx and Δy for the pull vector to be within $[-0.4, 0.4]$, rather than $[-1, 1]$. Restricting the action space may make it easier to learn visual dynamics models since larger deltas drastically alter the shape of the fabric.

We use the open-source software Blender to render (top-down) image observations $\mathbf{o}_t$ of the fabric. We make several changes for the observations relative to [184]. First, we use four-channel images: three for RGB, and one for depth. Second, we reduce the size of observations to 56×56 from 100×100, to make it more computationally tractable to train visual dynamics models. Finally, to enable transfer of policies trained in simulation to the real-world, we adjust the domain randomization [207] techniques so that color, brightness, and positional hyperparameters are fixed *per episode* to ensure that the video prediction model learns to only focus on predicting changes in the fabric configuration, rather than changes due to domain randomization. See Appendix C.3 for more details on the domain randomization parameters.

4.3.3 Data Generation

We collect 7,003 episodes of length 15 each for a total of 105,045 $(\mathbf{o}_t, \mathbf{a}_t, \mathbf{o}_{t+1})$ transitions for training the visuospatial dynamics model. The episode starting states are sampled from three tiers of difficulty with equal probability. These tiers are the same as those in [184], and largely based on *coverage*, or the amount that a fabric covers its underlying plane:

- **Tier 1 (Easy): High Coverage.** $78.3 \pm 6.9\%$ initial coverage, all corners visible. Generated by two short random actions.
- **Tier 2 (Medium): Medium Coverage.** $57.6 \pm 6.1\%$ initial coverage, one corner occluded. Generated by a vertical drop followed by two actions to hide a corner.
- **Tier 3 (Hard): Low Coverage.** $41.1 \pm 3.4\%$ initial coverage, 1-2 corners occluded. Generated by executing one action very high in the air and dropping.

All data was generated using the following policy: execute a randomly sampled action, but resample if the grasp point (x, y) is not within the bounding box of the 2D projection of the fabric, and take the additive inverse of Δx and/or Δy if $(x + \Delta x, y + \Delta y)$ is out of bounds (off the plane by more than 20% of its width, causing episode termination).

4.3.4 VisuoSpatial Dynamics Model

Due to the inherent stochasticity in fabric dynamics, we use Stochastic Variational Video Prediction (SV2P) from [7], a state-of-the-art method for action-conditioned video prediction. Here, the video prediction model with parameters θ is designed as a latent variable model, enabling the resulting posterior distribution to capture different modes in the distribution of predicted frames. Precisely, [7] trains a generative model which predicts a sequence of H output frames conditioned on a context vector of m frames and a sequence of actions starting from the most recent context frame. Since the stochasticity in video prediction is often a consequence of latent events not directly observable in the context frames as noted in [7], predictions are conditioned on a vector of latent variables $\mathbf{z}_{t+m:t+m+H-1}$, each sampled

from a fixed prior distribution $p(\mathbf{z})$. See [7] for more details on the model architecture and training procedure. For this work, we utilize the SV2P implementation provided in [211]. This gives rise to the following generative model.

$$p_\theta(\hat{\mathbf{o}}_{t+m:t+m+H-1}|\hat{\mathbf{a}}_{t+m-1:t+m+H-2}, \mathbf{o}_{t:t+m-1}) = \\ p(\mathbf{z}_{t+m}) \prod_{t'=t+m}^{t+m+H-1} p_\theta(\hat{\mathbf{o}}_{t'}|\mathbf{o}_{t:t+m-1}, \hat{\mathbf{o}}_{t+m:t'-1}, \mathbf{z}_{t'}, \hat{\mathbf{a}}_{t'-1}). \tag{4.4}$$

Here $\mathbf{o}_{t:t+m-1}$ are image observations from time t to $t+m-1$, $\hat{\mathbf{a}}_{t+m-1:t+m+H-2}$ is a candidate action sequence at timestep $t+m-1$, and $\hat{\mathbf{o}}_{t+m:t+m+H-1}$ is the sequence of predicted images. Since the generative model is trained in a recurrent fashion, it can be used to sample an H-length sequence of predicted images $\hat{\mathbf{o}}_{t+m:t+m+H-1}$ for any $m > 0, H > 0$ conditioned on a current sequence of image observations $\mathbf{o}_{t:t+m-1}$ and an H-length sequence of proposed actions taken from $\mathbf{o}_{t+m-1}$, given by $\hat{\mathbf{a}}_{t+m-1:t+m+H-2}$.

During training, we set the number of context frames $m = 3$ and number of output frames $H = 7$ (the SV2P model learns to predict 7 frames of an episode from the preceding 3 frames). We train on domain-randomized RGBD data, where we randomize fabric color (in a range centered around blue), shading of the plane, image brightness, and camera pose. At test time, we utilize only one context frame $m = 1$ and a planning horizon of $H = 5$ output frames. This yields the model $p_\theta(\hat{\mathbf{o}}_{t+1:t+5}|\hat{\mathbf{a}}_{t:t+4}, \mathbf{o}_t)$.

4.3.5 VisuoSpatial Foresight

To plan over the learned visuospatial dynamics model, we leverage the Cross-Entropy Method (CEM) [171] as in [38]. CEM repeatedly samples a population of action sequences from a multivariate Gaussian distribution, uses the learned dynamics model to predict output frames for each action sequence, identifies a set of "elite" sequences with the lowest cost according to the cost function in Section 4.3.1, and refits the Gaussian distribution to these elite sequences. We run 10 iterations of CEM with a population size of 2000 action sequences, 400 elites per iteration, and a planning horizon of 5. See Appendix C.2.3 for additional hyperparameters. Finally, to mitigate compounding model error we leverage Model-Predictive Control (MPC), which replans at each step and executes only the first action in the lowest cost action sequence found by CEM.

4.4 Simulated Experiments

4.4.1 VisuoSpatial Dynamics Prediction Quality

One advantage of VSF, and visual foresight more generally, is that we can inspect the model to see if its predictions are accurate, which provides some interpretability. Figure 4.3 shows three examples of predicted image sequences within episodes compared to ground truth

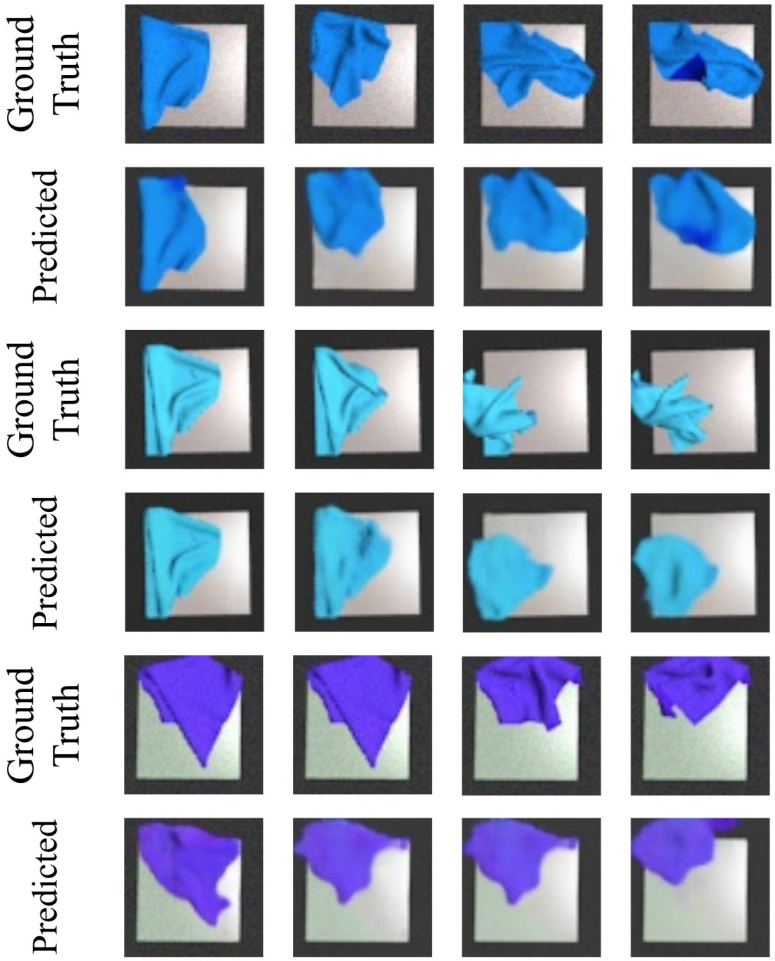

Figure 4.3: Three pairs of a sequence of four simulated image observations in ground truth compared against a sequence of the corresponding predictions from the visuospatial dynamics model. Here we show the RGB portion of the observations. Each episode is a test example and has randomized camera angle, fabric color, brightness, and shading. Prediction quality varies and generally gets blurrier across time, but is sufficient for planning.

images from the actual episodes. All episodes are from test time rollouts of a random policy, meaning that the ground truth images did not appear in the training data for the visuospatial dynamics model.

The predictions are reasonably robust to domain randomization, as the model is able to produce fabrics of roughly the appropriate color, and can appropriately align the angle of the light gray background plane. For a more quantitative measure of prediction quality, we calculate the average Structural SIMilarity (SSIM) index [218] over corresponding image pairs in 200 predicted sequences against 200 ground truth sequences to be 0.701. A higher

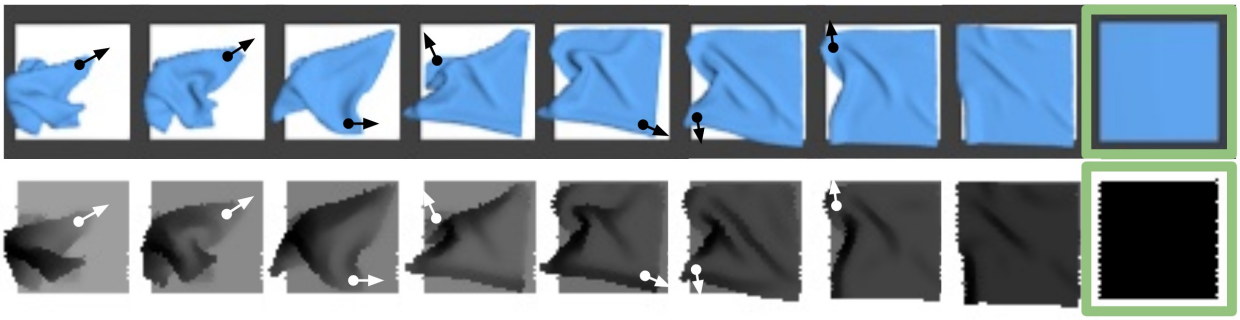

Figure 4.4: A simulated episode executed by the VisuoSpatial Foresight policy on a Tier 3 starting state, given a smooth goal image (shown in the far right). The first row shows RGB images and the second shows the corresponding depth maps. In this example, the policy is able to successfully cross the coverage threshold of 92% after executing 7 actions. Actions are visualized with the overlaid arrows.

SSIM $\in [-1, 1]$ corresponds to higher image similarity. SSIM $\in [-1, 1]$ has been shown in [218] to outperform similar metrics on evaluating prediction accuracy.

4.4.2 Fabric Smoothing Simulations

We first evaluate VSF on the smoothing task: maximizing fabric coverage, defined as the percentage of an underlying plane covered by the fabric. The plane is the same area as the fully smoothed fabric. We evaluate smoothing on three tiers of difficulty as reviewed in Section 4.3.3. Each episode lasts up to $T = 15$ time steps. Following [184], episodes can terminate earlier if a threshold of 92% coverage is triggered, or if any fabric point falls sufficiently outside of the fabric plane.

To see how the general VisuoSpatial Foresight policy performs against existing smoothing techniques, for each difficulty tier, we execute 200 episodes of VisuoSpatial Foresight (trained on random, domain-randomized RGBD data) with L2 cost and 200 episodes of each baseline policy in simulation. The L2 cost is taken with respect to a smooth goal image (see Figure 4.4). Note that VSF does *not* explicitly optimize for coverage, and only optimizes the cost function from Equation 4.2, which measures L2 pixel distance to a target image. We compare with the following 5 baselines:

Random Random pick point and pull direction.

Highest Using ground truth state information, pick the point mass with the maximum z-coordinate and set pull direction to point to where the point mass would be if the fabric were perfectly smooth. This is straightforward to implement with depth sensing and was shown to work reasonably well for smoothing in [185].

Wrinkle As in [198], find the largest wrinkle and then pull perpendicular to it at the edge of the fabric to smooth it out. We use the ground truth state information in the implementation of this algorithm (as done in [184]) rather than image observations.

Imitation Learning (IL) As in Seita et al. [184], train an imitation learning agent using DAgger [170] with a simulated "corner-pulling" demonstrator that picks and pulls at the fabric corner furthest from its target. DAgger can be considered as an oracle with "privileged" information as in [25] because during training, it queries a demonstrator which uses ground truth state information. For fair comparison, we run DAgger so that it consumes roughly the same number of data points (we used 110,000) as VisuoSpatial Foresight during training, and we give the policy access to four-channel RGBD images. We emphasize that this is a distinct dataset from the one used for VSF, which uses no demonstrations.

Model-Free RL We run DDPG [125] and extend it to use demonstrations and a pre-training phase as suggested in [213]. We also use the Q-filter from Nair et al. [148]. We train with a similar number of data points as in IL and VSF for a reasonable comparison. We design a reward function for the smoothing task that, at each time step, provides reward equal to the change in coverage between two consecutive states. Inspired by OpenAI et al. [152], we provide a $+5$ bonus for triggering a coverage success, and -5 penalty for pulling the fabric out of bounds.

Further details about the implementation and training of IL, DDPG, and VSF are in Appendix C.2.

Table 4.1 indicates that VSF significantly outperforms the analytic and model-free reinforcement learning baselines. It has similar performance to the IL agent, a "smoothing specialist" that rivals the performance of the corner pulling demonstrator used in training (see Appendix C.2). See Figure 4.4 for an example Tier 3 VSF episode. Furthermore, we find coverage values to be statistically significant compared to all baselines other than IL and not much impacted by the presence of domain randomization. Results from the Mann-Whitney U test [137] and a domain randomization ablation study are reported in Appendix C.3. VSF, however, requires more actions than DAgger, especially on Tier 1, with 8.3 actions per episode compared to 3.3 per episode. We hypothesize that this leads to VSF being more conservative and reliable, which is supported by VSF having lower variance in performance on Tier 2 and Tier 3 settings (Table 4.1).

4.4.3 Fabric Folding Simulations

Here we demonstrate that VSF is not only able to smooth fabric, but also directly generalizes to folding tasks. We use the same video prediction model, trained only with random interaction data, and keep planning parameters the same besides the initial CEM variance (see Appendix C.2.3). We change the goal image to the triangular, folded shape shown in

Table 4.1: Simulated smoothing experimental results for the six policies in Section 4.4.2. We report final coverage and number of actions per episode, averaged over 200 simulated episodes per tier. VisuoSpatial Foresight (VSF) performs well even for difficult starting states. It attains similar final coverage as the Imitation Learning (IL) agent from [184] and outperforms the other baselines. The VSF and IL agents were trained on equal amounts of domain-randomized RGBD data, but the IL agent has a demonstrator for every training state, whereas VSF is trained with data collected from a random policy.

Tier	**Method**	**Coverage**	**Actions**
1	Random	25.0 ± 14.6	2.4 ± 2.2
1	Highest	66.2 ± 25.1	8.2 ± 3.2
1	Wrinkle	91.3 ± 7.1	5.4 ± 3.7
1	DDPG and Demos	87.1 ± 10.7	8.7 ± 6.1
1	Imitation Learning	94.3 ± 2.3	3.3 ± 3.1
1	VisuoSpatial Foresight	92.5 ± 2.5	8.3 ± 4.7
2	Random	22.3 ± 12.7	3.0 ± 2.5
2	Highest	57.3 ± 13.0	10.0 ± 0.3
2	Wrinkle	87.0 ± 10.8	7.6 ± 2.8
2	DDPG and Demos	82.0 ± 14.7	9.5 ± 5.8
2	Imitation Learning	92.8 ± 7.0	5.7 ± 4.0
2	VisuoSpatial Foresight	90.3 ± 3.8	12.1 ± 3.4
3	Random	20.6 ± 12.3	3.8 ± 2.8
3	Highest	36.3 ± 16.3	7.9 ± 3.2
3	Wrinkle	73.6 ± 19.0	8.9 ± 2.0
3	DDPG and Demos	67.9 ± 15.6	12.9 ± 3.9
3	Imitation Learning	88.6 ± 11.5	10.1 ± 3.9
3	VisuoSpatial Foresight	89.3 ± 5.9	13.1 ± 2.9

Figure 4.5 and change the initial state to a smooth state (which can be interpreted as the result of a smoothing policy from Section 4.4.2). The two sides of the fabric are shaded differently, with the darker shade on the bottom layer. Due to the action space bounds, getting to this goal state directly is not possible in under three actions and requires a precise sequence of pick and pull actions.

We visually inspect the final states in each episode, and classify them as successes or failures. For RGBD images, this decision boundary empirically corresponds to an L2 threshold of roughly 2500; see Figure 4.5 for a typical success case. In Table 4.2 we compare performance of L2 cost taken over RGB, depth, and RGBD channels. RGBD significantly outperforms the other modes, which correspond to Visual Foresight and "Spatial Foresight" (*i.e.,* depth only), suggesting the usefulness of augmenting Visual Foresight with depth maps. For a more complete analysis of the impact of input modality, we also provide a histogram of coverage values obtained on the simulated *smoothing* task for RGB, D, and RGBD in

Table 4.2: Single folding results in simulation. VisuoSpatial Foresight is run with the goal image in Figure 4.5 for 20 episodes when L2 is taken on the depth, RGB, and RGBD channels. The results suggest that adding depth allows us to significantly outperform RGB-only Visual Foresight on this task.

Cost Function	Successes	Failures	% Success
L2 Depth	0	20	0%
L2 RGB	10	10	50%
L2 RGBD	**18**	**2**	**90%**

Figure 4.7. Here RGBD also performs the best but only slightly outperforms RGB, which is perhaps unsurprising due to the relatively low depth variation in the smoothing task.

We then attempt to reach a more complex goal image with two overlapping folds for a total of three layers at the center of the fabric, again with the same VSF policy. Here 8 out of 20 trials succeed, and 7 of the 8 successes arrange the two folds in the correct ordering. Folding two opposite corners in the right order may be difficult to do solely with RGB data, since the bottom layer of the fabric has a uniform color. The depth maps, in contrast, provide information about one fold lying on top of another. See Figure 4.6 for an example of a successful rollout. Failure cases are often characterized by the robot picking at a point that just misses a fabric corner, but where the visual dynamics model predicts that the resulting pull will behave as if the robot had picked at the fabric corner.

4.5 Physical Experiments

We next deploy the domain randomized policies on a physical da Vinci surgical robot [97]. We use the same experimental setup as in Seita et al. [184], including the calibration procedure to map pick points (x, y) into positions and orientations with respect to the robot's base frame. The sequential tasks we consider are challenging due to the robot's imprecision [186]. We use a Zivid One Plus camera mounted 0.9 meters above the workspace to capture RGBD images.

4.5.1 Experiment Protocol

We evaluate the best Imitation Learning and the best VisuoSpatial Foresight policies as measured in simulation and reported in Table 3.1. We do not test with the model-free DDPG policy baseline, as it performed significantly worse than the other two methods. For IL, this is the final model trained with 110,000 actions based on a corner-pulling demonstrator with access to state information. This uses slightly more than the 105,045 actions used for training the VSF model. We reiterate that VSF uses a video prediction model that is trained on *entirely random data*, requiring no labeled data in contrast with IL, which uses DAgger and requires a demonstrator with "privileged" fabric state information during training. The

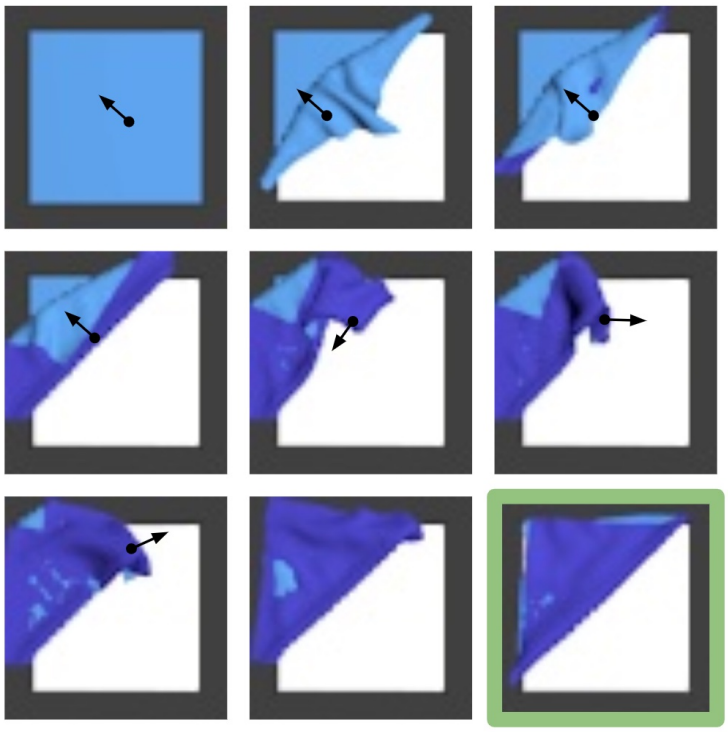

Figure 4.5: Simulated RGB observations of a successful folding episode. The RGB portion of the goal image is displayed in the bottom right corner. The rest is an 8-step episode (left-to-right, top-to-bottom) from smooth to approximately folded. There are several areas of the fabric simulator which have overlapping layers due to the difficulty of accurately modeling fabric-fabric collisions in simulation, which explain the light blue patches in the figure.

demonstrator uses state information to determine the fabric corner furthest from its target on the underlying plane, and pulls the fabric at that fabric corner to the target.

To match the simulation setup, we limit each episode to a maximum of 15 actions. For both methods on the *smoothing* task, we perform episodes in which the fabric is initialized in highly rumpled states which mirror those from the simulated tiers. We run ten episodes per tier for IL and five episodes per tier for VSF, for 45 episodes in all. In addition, within each tier, we attempt to make starting fabric states reasonably comparable among IL and VSF episodes (e.g., see Figure 4.8).

4.5.2 Physical Fabric Smoothing

We present quantitative results in Table 4.3 that suggest that VSF gets final coverage results comparable to that of IL, despite not being trained on any corner-pulling demonstration data. However, it sometimes requires more actions to complete an episode and takes significantly

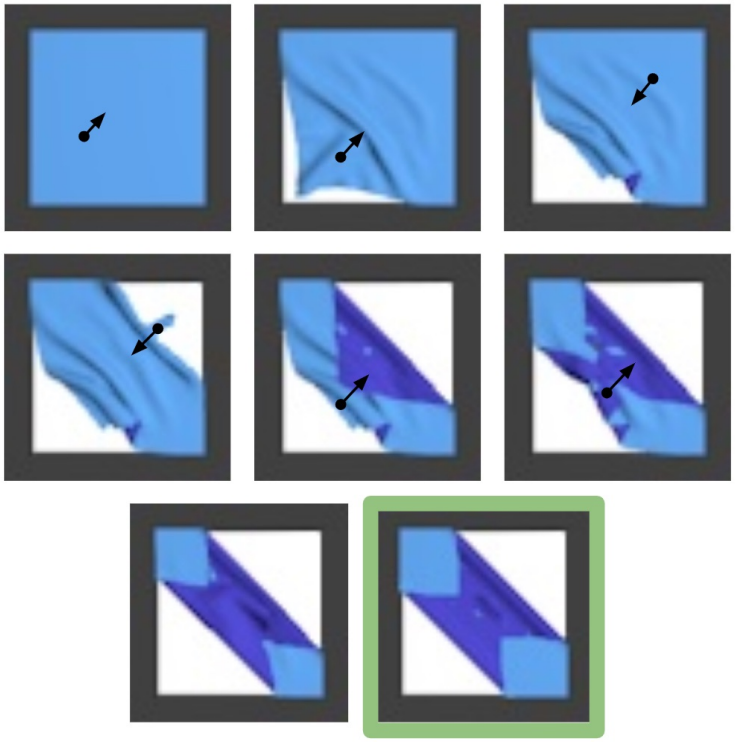

Figure 4.6: An example of a successful double folding episode. The RGB portion of the goal image is displayed in the bottom row. The rest is an 7-step trajectory (left-to-right, top-to-bottom) from smooth to approximately matching the goal image. The two folds are stacked in the correct order.

more computation time, as the Cross Entropy Method requires thousands of forward passes through a large network while IL requires only a single pass. The actions for VSF usually have smaller deltas, which helps to take more precise actions. This is likely due in part to the initialization of the mean and variance used to fit the conditional Gaussians for CEM. Figure C.4 in Appendix C.3.3 shows histograms of the action delta magnitudes. The higher magnitude actions of the IL policy may cause it to be more susceptible to highly counter-productive actions. The fabric manipulation tasks we consider require high precision, and a small error in the pick point region coupled with a long pull may cover a corner or substantially decrease coverage.

As an example, Figure 4.8 shows a time lapse of a subset of actions for one episode from IL and VSF. Both begin with a fabric of roughly the same shape to facilitate comparisons. On the fifth action, the IL policy has a pick point that is slightly north of the ideal spot. The pull direction to the lower right fabric plane corner is reasonable, but due to the length of the pull, combined with a slightly suboptimal pick point, the lower right fabric corner gets covered. This makes it harder for a policy trained from a corner-pulling demonstrator to get

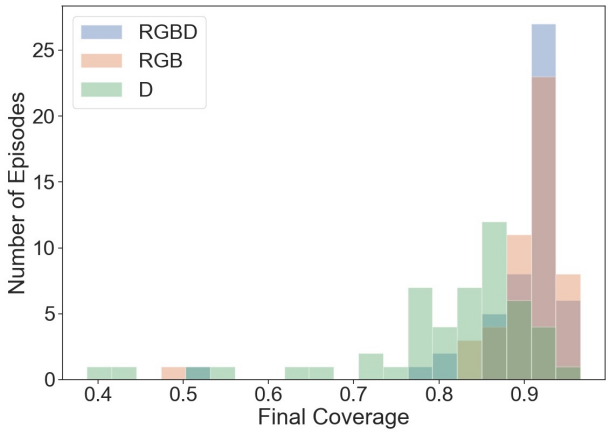

Figure 4.7: Final coverage values on 50 simulated smoothing episodes from Tier 3 starting states. We fix the random seed so that each input modality (RGB, D and RGBD) begins with the same starting states.

Table 4.3: Physical smoothing robot experiment results for Imitation Learning (IL), i.e. DAgger, and VisuoSpatial Foresight (VSF). For both methods, we choose the policy snapshot with highest performance in simulation, and each are applied on all tiers (T1, T2, T3). We show results across 10 episodes of IL per tier and 5 episodes of VSF per tier, and show average starting and final coverage, maximum coverage at any point per episode, and the number of actions. Results suggest that VSF attains final coverage comparable to or exceeding that of IL despite not being trained on demonstration data, though VSF requires more actions per episode.

Tier	**(1) Start**	**(2) Final**	**(3) Max**	**(4) Actions**
1 IL	74.2 ± 5	92.1 ± 6	92.9 ± 3	4.0 ± 3
1 VSF	78.3 ± 6	**93.4 ± 2**	93.4 ± 2	8.2 ± 4
2 IL	58.2 ± 3	84.2 ± 18	86.8 ± 15	9.8 ± 5
2 VSF	59.5 ± 3	**87.1 ± 9**	90.0 ± 5	12.8 ± 3
3 IL	43.3 ± 4	75.2 ± 18	79.1 ± 14	12.5 ± 4
3 VSF	41.4 ± 3	**75.6 ± 15**	76.9 ± 15	15.0 ± 0

high coverage, as the fabric corner is hidden. In contrast, the VSF policy takes actions of shorter magnitudes and does not fall into this trap. The downside of VSF, however, is that it may "waste" too many actions with short magnitudes, whereas IL can quickly get high coverage conditioned on accurate pick points. In future work, we will develop ways to tune VSF so that it takes longer actions as needed.

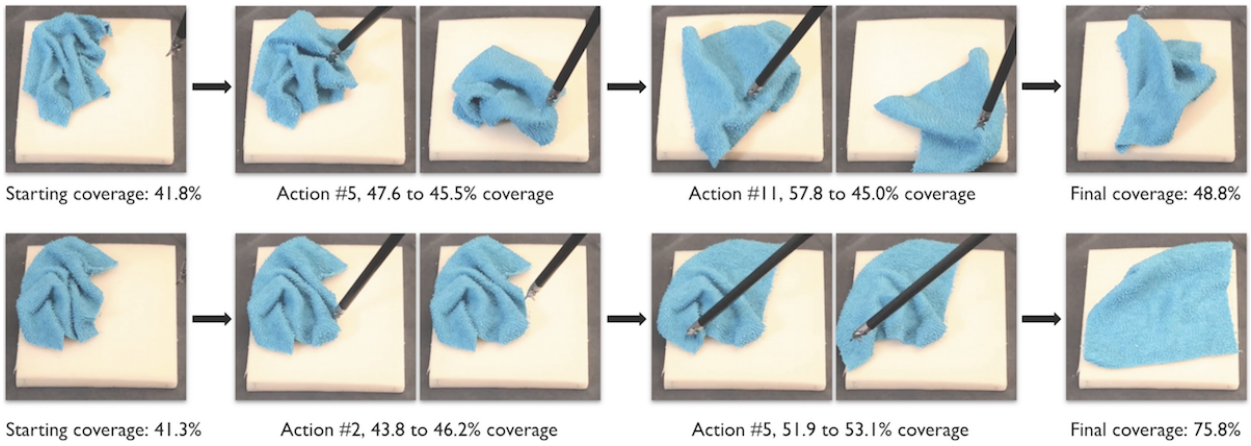

Figure 4.8: A qualitative comparison of physical da Vinci episodes with an Imitation Learning policy (top row) and a VisuoSpatial Foresight policy (bottom row). The rows show screen captures taken from the third-person video view for recording episodes; these are not the input to VisuoSpatial Foresight. To facilitate comparisons among IL and VSF, we manually make the starting fabric state as similar as possible. Over the course of several actions, the IL policy sometimes takes actions that are highly counter-productive, such as the 5th and 11th actions above. Both pick points are reasonably chosen, but the large deltas cause the lower right fabric corner to get hidden. In contrast, VSF, takes shorter pulls on average, with representative examples shown above for the 2nd and 5th actions. At the end, the IL policy gets just 48.8% coverage (far below its usual performance), whereas VSF gets 75.8%. For further quantitative results, see Table 4.3.

4.5.3 Physical Fabric Folding

We next evaluate the *same* VSF policy, but on a fabric *folding* task, starting from a smooth state. We conduct four goal-conditioned physical folding episodes, with one episode for each of the four possible diagonal folds. We run each episode for 15 time steps. Qualitatively, the robot tends to move the fabric in the right direction based on the target corner, but does not get the fabric in a clean two-layer fold. To evaluate the quality of the policy, we take the actions generated for a *real episode* on the physical system and run them open-loop in the fabric simulator. See Figure 4.9 for a comparison. Since the *same actions* are able to fold reasonably well in simulation, we conclude that the difference is due to a dynamics mismatch between simulated and real environments, compounded with pick point inaccuracies common in cable-driven robots such as the dVRK [186]. In future work, we will improve the simulator's physics to more closely match those of the dVRK, and we will also consider augmenting the video prediction model with a small amount of real-world data.

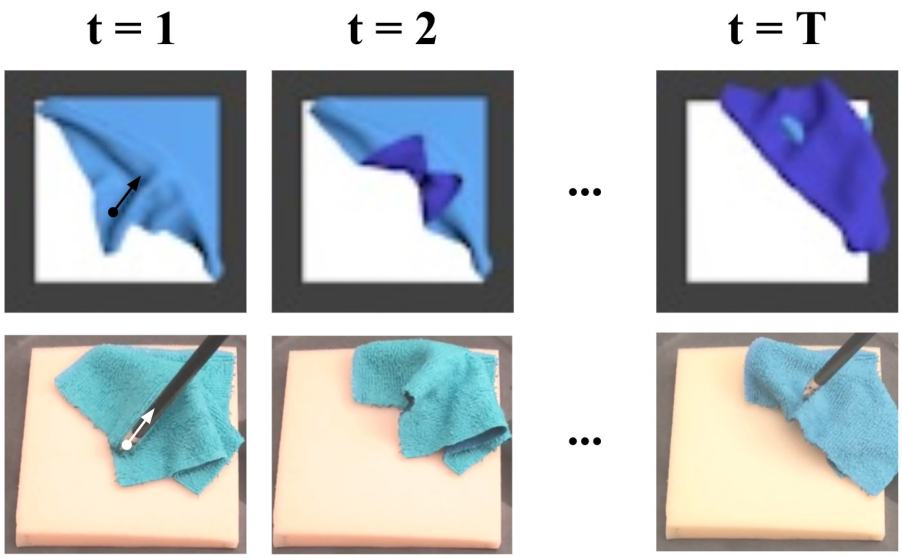

Figure 4.9: Physical folding policy executed in the simulator and on the surgical robot, with actions determined from *real* image input only. Despite this, the actions are able to fold in simulation. The difference in dynamics is apparent from $t = 1$ to $t = 2$, where the simulated fabric's bottom left corner is overturned from the action, but the corresponding corner on the real fabric is not.

4.6 Conclusion and Future Work

We present VisuoSpatial Foresight, which leverages a combination of RGB and depth information to learn goal conditioned fabric manipulation policies for a variety of long horizon tasks. We train a video prediction model on purely random interaction data with fabric in simulation, and demonstrate that planning over this model with MPC results in a policy with promising generalization across goal-directed fabric smoothing and folding tasks. We then transfer this policy to a real robot system with domain randomization. In future work, we will explore learned cost functions, test different fabric shapes, and investigate learning bilateral manipulation or human-in-the-loop policies.

Part III

Benchmarks and Architectures

Chapter 5

Learning to Rearrange Deformable Cables, Fabrics, and Bags with Goal-Conditioned Transporter Networks

Manipulating deformable objects is a long-standing challenge in robotics with a wide range of real-world applications. In contrast to rigid object manipulation, deformable object manipulation presents additional challenges due to more complex configuration spaces, dynamics, and sensing.

While several prior works have made progress largely centered around tasks with 1D (*e.g.,* rope [147]) or 2D (*e.g.,* fabric [184]) deformable structures, little prior work has addressed generalizable vision-based methods for manipulating 3D deformable structures such as the task of: "insert objects into a bag, and then carry the bag away". These types of tasks are especially challenging with diverse goal conditioning: the goal states of deformable objects are not easily specified, for example, by compact pose representations.

In this chapter, we propose a new suite of benchmark tasks, called *DeformableRavens*, to test manipulation of cables, fabrics, and bags spanning 1D, 2D, and 3D deformables. For several tasks in the benchmark, we propose to tackle them using novel goal-conditioned variants of Transporter Network [240] architectures, which enable Transporter Networks to be specified to achieve diverse goals. Our experiments also significantly extend results of using Transporter Networks for deformable manipulation tasks — while [240] demonstrated one 1D deformable task, we show results on 12 tasks, including those with fabrics and bags.

The main contributions of this chapter are: (i) an open-source simulated benchmark, *DeformableRavens*, with 12 tasks manipulating 1D, 2D, and 3D deformable objects to help accelerate research progress in robotic manipulation of deformables, (ii) end-to-end goal-conditioned Transporter Network architectures that learn vision-based multi-step manipulation of deformable 1D, 2D, and 3D structures, and (iii) experiments demonstrating that the proposed vision-based architectures are competitive with or superior to baselines that

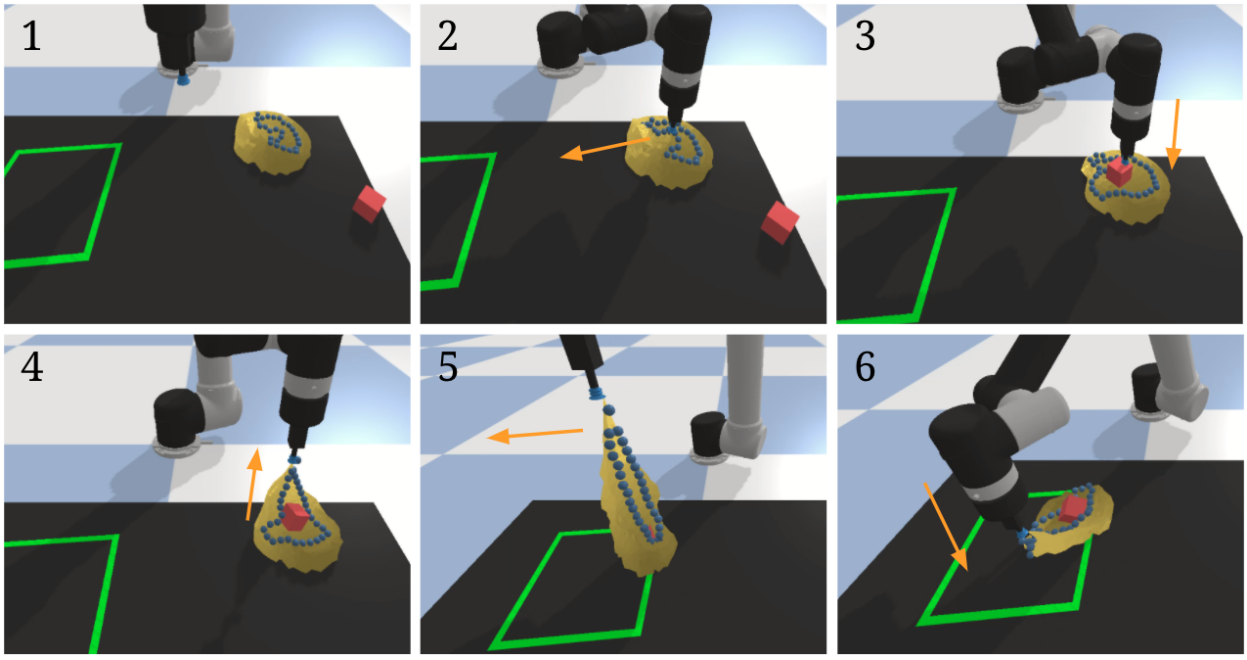

Figure 5.1: Example of a trained Transporter Network policy in action on the *bag-items-1* task (see Table 5.1). The setup involves a simulated UR5 robot arm, a green target zone, a drawstring-style bag, and a red cube. The starting configuration is shown in the top left frame. The robot, with suction cup, must grasp and sufficiently open the bag to insert a cube, then (bottom row) picks and pulls upwards, enclosing the cube. The robot concludes by bringing the bag with the cube in the target zone. We overlay arrows to indicate the movement of the robot's arm just before a given frame.

consume ground truth simulated pose and vertex information. We also discuss the shortcomings of the system, which point to interesting areas for future research. The project website contains supplementary material, including the appendix, code, data, and videos: `https://berkeleyautomation.github.io/bags/`.

5.1 Related Work

The present work builds upon an extensive literature involving deformable object manipulation [177], data-driven robot manipulation, and training image-based policies.

5.1.1 Deformable Object Manipulation

In this work, deformables [177] refers to 1D structures such as ropes and cables (we use the terms interchangeably), 2D structures such as fabrics, and 3D structures such as bags.

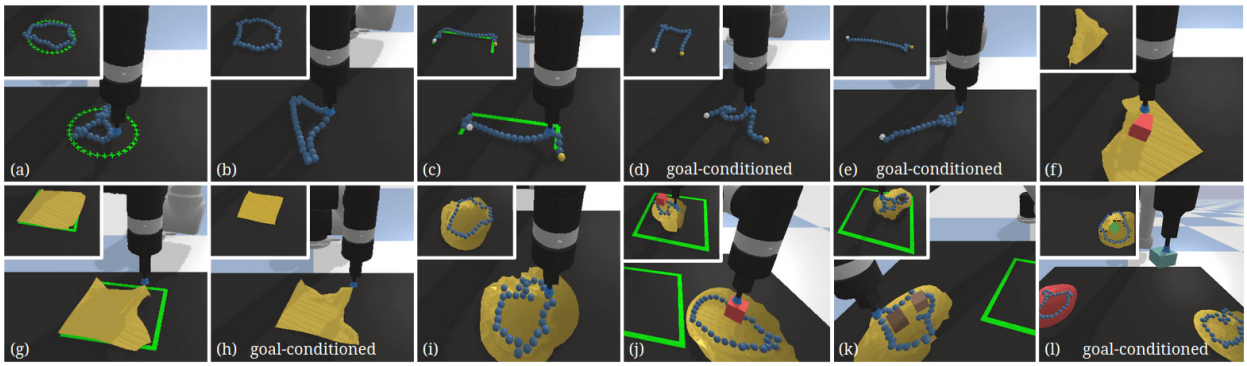

Figure 5.2: The 12 tasks in the proposed *DeformableRavens* benchmark (see Table 5.1) with suction cup gripper and deformable objects. **Top row**: (a) cable-ring, (b) cable-ring-notarget, (c) cable-shape, (d) cable-shape-notarget, (e) cable-line-notarget, (f) fabric-cover. **Bottom row**: (g) fabric-flat, (h) fabric-flat-notarget, (i) bag-alone-open, (j) bag-items-1, (k) bag-items-2, (l) bag-color-goal. Of the tasks shown, 4 are "goal-conditioned," *i.e.,* cable-shape-notarget, cable-line-notarget, fabric-flat-notarget, bag-color-goal. These use a separate goal image $\mathbf{o}_g$ for each episode to specify a success configuration. The other 8 tasks do not use goal images. Examples of successful configurations for each task are shown with overlaid (cropped) images to the top left.

Rope has been extensively explored in early robotics literature, such as for knot-tying [84, 145]. In recent years, learning-based approaches for manipulation have grown in popularity to improve the generalization of knot-tying and also to manipulate rope towards general target configurations [183, 147, 159, 199, 228, 217]. The approach we propose is focused primarily in manipulating cables to target locations, which may be specified by a target zone or a target image, and does not require modeling rope physics.

Much research on robotic fabric manipulation [21, 177, 29] uses bilateral robots and gravity to expose corners for ease of manipulation, which enables iteratively re-grasping the fabric's lowest hanging point [155, 104, 105]. Subsequent work [136, 31, 37], generalized to a wider variety of initial configurations of new fabrics. Other approaches focus on fabric smoothing. For example, [220] present an algorithm that pulls at eight fixed angles, and identifies corners using the Harris Corner Detector [71]. Sun et al. [197, 198] followed up by attempting to explicitly detect and then pull perpendicular to the largest wrinkle, and tested on fabrics in nearly planar configurations. Another task of interest is wrapping rigid objects with fabric, which bridges the gap between manipulating 2D and 3D structures. Hayashi et al. [74, 75] present an approach for wrapping fabric around a cylinder using a bimanual robot.

Work on 3D deformable structures such as bags has been limited due to the complexities of manipulating bags. Early work focused on mechanical design of robots suitable for grasping [98] or unloading [103] sacks. Some work assumes that a sturdy, brown bag is already open for item insertion, as with a grocery checkout robot [106], or uses reinforcement

learning for zipping bags [78] in constrained setups.

These works generally focus on optimizing a specific problem in deformable object manipulation, often with task-specific algorithms or manipulators. In contrast, we propose a simulation setup with a single robot arm that applies to a wide range of tasks with 1D, 2D, and 3D deformables. In independent and concurrent work, Lin et al. [128] propose *SoftGym*, a set of environments with cables, fabrics, and liquids simulated using NVIDIA FleX, and use it to benchmark standard reinforcement learning algorithms.

5.1.2 Data-Driven Robot Manipulation

Robot manipulation using learned, data-driven techniques has become a highly effective paradigm for robot manipulation, as exemplified in the pioneering work of Levine et al. [119], Mahler et al. [135], Florence et al. [51], Kalashnikov et al. [96], and others. Tools used often involve either Imitation Learning (IL) [6] or Reinforcement Learning (RL) [201], and in recent years, such techniques have been applied for manipulation of deformables.

For example, IL is used with human [185] and scripted [184] demonstrators for fabric smoothing, while RL is applied for smoothing and folding in [139, 93, 227]. Using model-based RL, [38] were able to train a video prediction model to enable robots to fold pants and fabric, and a similar approach was investigated further in [85] for model-based fabric manipulation. Some approaches combine IL and RL [9], and others use techniques such as latent space planning [129, 229] or using dense object descriptors [61]. In this work, we use IL.

As described in Section 5.2, the deep architecture we propose for image-based manipulation uses Fully Convolutional Neural Networks (FCNs) [130] to produce per-pixel scores in an image, where each pixel corresponds to an action. This technique [236] has shown promising results for robotic warehouse order picking [238], pushing and grasping [237], tossing objects [239], kit assembly [235], and mobile manipulation [223]. We show that similar approaches may be effective for picking and placing of deformables. In independent and concurrent work, Lee et al. [118] show how an approach can be used for fabric folding.

5.2 Background

We first describe the problem formulation, followed by background on Transporter Networks [240]. Section 5.3 then describes a novel goal-conditioned framework.

5.2.1 Problem Formulation

We formulate the problem of rearranging deformable objects as learning a policy π that sequences pick and place actions $\mathbf{a}_t \in \mathcal{A}$ with a robot from visual observations $\mathbf{o}_t \in \mathcal{O}$:

$$\pi(\mathbf{o}_t) \rightarrow \mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}) \in \mathcal{A} \tag{5.1}$$

where $\mathcal{T}_{\text{pick}}$ is the pose of the end effector when grasping part of an object, and $\mathcal{T}_{\text{place}}$ is the pose of the end effector when releasing the grasp. Since deformable objects may require more than a single pick and place action to reach a desired configuration, it is important that the policy π learns from closed-loop visual feedback. Some rearrangement tasks [13] may also be specified by a target goal observation $\mathbf{o}_g$, in which case we refer to the policy as goal-conditioned:

$$\pi(\mathbf{o}_t, \mathbf{o}_g) \rightarrow \mathbf{a}_t = (\mathcal{T}_{\text{pick}}, \mathcal{T}_{\text{place}}) \in \mathcal{A}. \tag{5.2}$$

In this work, we consider tabletop manipulation tasks where both poses $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$ are defined in SE(2). Picking and placing positions are sampled from a fixed-height planar surface, while rotations are defined around the z-axis (aligned with gravity direction). The $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$ both parameterize a motion primitive [54] that controls the end effector to approach $\mathcal{T}_{\text{pick}}$ until contact is detected, activates the end effector to execute a grasp, moves upward to a fixed z-height, approaches $\mathcal{T}_{\text{place}}$, and lowers the end effector until contact is detected, then releases the grasp. While this discrete-time planar action parameterization has its limitations, we find that it remains sufficiently expressive for a number of tabletop tasks involving manipulation of bags, in which gravity helps to induce a natural adhesive force that brings objects back towards the tabletop plane.

To train the policy, we assume access to a small dataset of N stochastic expert demonstrations $\mathcal{D} = \{\xi_i\}_{i=1}^{N}$, where each *episode* ξ_i of length T_i (*i.e.,* the number of actions) consists of a sequence of observations and actions:

$$\xi_i = (\mathbf{o}_1, \mathbf{a}_1, \mathbf{o}_2, \ldots, \mathbf{o}_{T_i}, \mathbf{a}_{T_i}, \mathbf{o}_{T_i+1}) \tag{5.3}$$

used to supervise π, which may be goal-conditioned.

5.2.2 Background: Transporter Networks

Transporter Networks [240] is a model architecture for manipulation that learns to rearrange objects by (i) attending to a local region of interest, then (ii) predicting its target spatial displacement by cross-correlating its dense deep visual features over the scene. It can parameterize actions for pick and place, and has shown to work well for variety of tabletop manipulation tasks including stacking, sweeping, and kit assembly [235].

Transporter Networks consist of 3 Fully Convolutional Networks (FCNs). The first FCN f_{pick} takes as input the visual observation $\mathbf{o}_t$, and outputs a dense per-pixel prediction of action-values $\mathcal{Q}_{\text{pick}}$ that correlate with picking success: $\mathcal{T}_{\text{pick}} = \arg\max_{(u,v)} \mathcal{Q}_{\text{pick}}((u,v)|\mathbf{o}_t)$ where each pixel (u, v) corresponds to a picking action at that location via camera-to-robot calibration. The second FCN Φ_{key} also takes as input $\mathbf{o}_t$, while the third FCN Φ_{query} takes as input a partial crop $\mathbf{o}_t[\mathcal{T}_{\text{pick}}]$ from $\mathbf{o}_t$ centered on $\mathcal{T}_{\text{pick}}$. Both the second and third FCNs output dense feature embeddings, which are then cross-correlated with each other to output a dense per-pixel prediction of action-values $\mathcal{Q}_{\text{place}}$ that correlate with placing success:

$$\mathcal{T}_{\text{place}} = \arg\max_{\{\Delta\tau_i\}} \mathcal{Q}_{\text{place}}(\Delta\tau_i|\mathbf{o}_t, \mathcal{T}_{\text{pick}}) \tag{5.4}$$

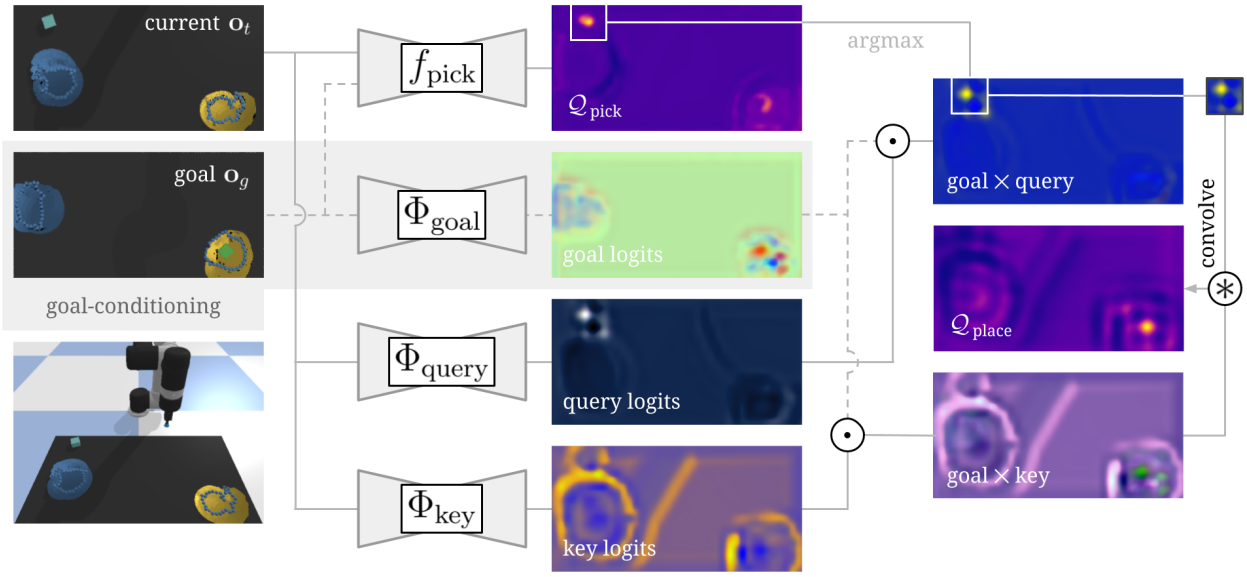

Figure 5.3: The proposed *Transporter-Goal-Split*, applied to an example on *bag-color-goal*, where given the current image $\mathbf{o}_t$ and goal $\mathbf{o}_g$, the objective is to insert the block in the correct, open bag; the goal $\mathbf{o}_g$ specifies that the yellow bag (with the block) is the target, not the blue bag. Transporter Networks use three Fully Convolutional Networks (FCNs), f_{pick} for the attention module, and Φ_{query} and Φ_{key} for the transport module. *Transporter-Goal-Split* considers a separate goal image $\mathbf{o}_g$ and passes that through a new FCN, Φ_{goal}. These are then combined with other deep features (labeled as "logits") through element-wise multiplication. Finally, the standard cross-convolution operator is applied. *Transporter-Goal-Stack* dispenses with Φ_{goal} and instead stacks $\mathbf{o}_t$ and $\mathbf{o}_g$ as one image, and does not use Φ_{goal}). All tested Transporter architectures produce $\mathcal{Q}_{\text{pick}}$ and $\mathcal{Q}_{\text{place}}$, which are each 320×160 dimensional heat maps colored so that darker pixels are low values and lighter pixels are high. The largest value in a heat map represents the *pixel* to pick or place.

where

$$\mathcal{Q}_{\text{place}}(\Delta\tau|\mathbf{o}_t, \mathcal{T}_{\text{pick}}) = \Phi_{\text{query}}(\mathbf{o}_t[\mathcal{T}_{\text{pick}}]) * \Phi_{\text{key}}(\mathbf{o}_t)[\Delta\tau] \tag{5.5}$$

and $\Delta\tau$ covers the space of all possible placing poses.

A key aspect of Transporter Networks is that the visual observations $\mathbf{o}_t$ must be spatially consistent, so that the 3D structure of the data is preserved under different visuo-spatial transforms $\Delta\tau$. In practice, this property also improves training data augmentation (since rotations and translations of the orthographic image appear as different configurations of objects in the scene). To leverage this, observations $\mathbf{o}_t$ are top-down orthographic images of the tabletop scene, where each pixel represents a vertical column of 3D space.

5.3 Goal-Conditioned Transporter Networks

In some tasks, it is more natural to specify a success criteria by passing in a goal observation $\mathbf{o}_g$ showing objects in a desired configuration. In this case, we assume $\mathbf{o}_g$ is fixed and available as an extra input with the current observation $\mathbf{o}_t$, and that $\mathbf{o}_t$ and $\mathbf{o}_g$ have the same pixel resolution.

5.3.1 Goal-Conditioned Transporter Networks

We propose two goal-conditioned architectures based on Transporter Networks. The first, *Transporter-Goal-Stack*, stacks the current $\mathbf{o}_t$ and goal $\mathbf{o}_g$ images channel-wise, then passes it as input through a standard Transporter Network. The second, *Transporter-Goal-Split*, separates processing of the goal image $\mathbf{o}_g$ through a fourth FCN Φ_{goal} to output dense features that are combined with features from the query Φ_{query} and key Φ_{key} networks using the Hadamard product:

$$\psi_{\text{query}}(\mathbf{o}_t) = \Phi_{\text{query}}(\mathbf{o}_t) \odot \Phi_{\text{goal}}(\mathbf{o}_t) \tag{5.6}$$

$$\psi_{\text{key}}(\mathbf{o}_t) = \Phi_{\text{key}}(\mathbf{o}_t) \odot \Phi_{\text{goal}}(\mathbf{o}_t) \tag{5.7}$$

$$\mathcal{Q}_{\text{place}}(\Delta\tau|\mathbf{o}_t, \mathcal{T}_{\text{pick}}) = \psi_{\text{query}}(\mathbf{o}_t)[\mathcal{T}_{\text{pick}}] * \psi_{\text{key}}(\mathbf{o}_t)[\Delta\tau] \tag{5.8}$$

We hypothesize that this separation is beneficial for learning, since convolutional kernels may otherwise struggle to disambiguate which channels correspond to $\mathbf{o}_t$ or $\mathbf{o}_g$. Figure 5.3 shows the architecture of *Transporter-Goal-Split*. For consistency, all Transporter modules Φ_{query}, Φ_{key}, and Φ_{goal} employ a 43-layer, 9.9M parameter FCN [130] with residual connections [76], given that similar architectures have been used for FCN-based picking and placing [238, 240].

A key aspect of the goal-conditioned model is that the spatial structure of the goal images are preserved in the architecture. This prior encourages the deep networks to learn features that infer pick and place actions anchored on visual correspondences between objects in the current and goal images. This is useful for manipulating deformable objects, where changes between current and desired configurations are better captured with dense correspondences [61, 51].

5.3.2 Training Details for Goal Conditioned Models

To train both goal-conditioned architectures, we use an approach based on Hindsight Experience Replay [4]. For each task (Table 5.1), we assume a dataset of N demonstrations $\mathcal{D} = \{\xi_i\}_{i=1}^{N}$, where each episode ξ_i (see Eq. 5.3) is of length T_i with final observation $\mathbf{o}_g = \mathbf{o}_{T_i+1}$. In standard Transporter Networks, a single sample $(\mathbf{o}_k, \mathbf{a}_k)$ containing the observation and the resulting action at time k, is drawn at random from $\mathcal{D}$. For the goal-conditioned Transporter Networks, we use the same procedure to get $(\mathbf{o}_k, \mathbf{a}_k)$, then additionally use the corresponding observation $\mathbf{o}_g$ after the last action from the demonstration episode containing $\mathbf{o}_k$, to get training sample $(\mathbf{o}_k, \mathbf{a}_k, \mathbf{o}_g)$.

Table 5.1: **DeformableRavens**. Tasks involve rearranging deformable objects (*e.g.,* cables, fabrics, and bags). Each comes with a scripted expert demonstrator that succeeds with high probability, except for the four bag tasks which are challenging; for these, we filter to use only successful episodes in training. Some require *precise placing* to trigger a success. Tasks with a *visible zone* will have a green target zone on the workspace to indicate where items should be placed (e.g., a square target zone that a fabric must cover); other *goal-conditioned* tasks use a separate goal image to specify the success criteria for object rearrangement. See Figure 5.2 for visualizations.

Task (Max. Episode Length)	demos stats(%)	precise placing	visible zone	goal cond.
(a) cable-ring[§] (20)	99.1	✗	✓	✗
(b) cable-ring-notarget[§] (20)	99.3	✗	✗	✗
(c) cable-shape* (20)	98.8	✓	✓	✗
(d) cable-shape-notarget* (20)	99.1	✓	✗	✓
(e) cable-line-notarget* (20)	100.0	✓	✗	✓
(f) fabric-cover (2)	97.0	✗	✗	✗
(g) fabric-flat[†] (10)	98.3	✓	✓	✗
(h) fabric-flat-notarget[†] (10)	97.4	✓	✗	✓
(i) bag-alone-open[§] (8)	60.2	✗	✗	✗
(j) bag-items-1 (8)	41.7	✗	✓	✗
(k) bag-items-2 (9)	32.5	✗	✓	✗
(l) bag-color-goal (8)	89.1	✗	✗	✓

[§]evaluated based on maximizing the convex hull area of a ring.
*evaluated by the percentage of a cable within a target zone.
[†]evaluated using fabric coverage, as in Seita et al. [184, 185].

As described in Section 5.2.2, because observations $\mathbf{o}_t$ are top-down orthographic images and are spatially consistent with rotations and translations, this enables data augmentation by randomizing a rotation and translation for each training image. To handle the goal-conditioned case, both $\mathbf{o}_t$ and $\mathbf{o}_g$ are augmented using the same random rotation and translation, to ensure consistency.

5.4 Simulator and Tasks

We evaluate the system on *DeformableRavens*, a novel suite of simulated manipulation tasks involving cables, fabrics, and bags, using PyBullet [30] with an OpenAI gym [22] interface. See Table 5.1 and Figure 5.2 for overviews.

5.4.1 Deformable Objects (Soft Bodies) in PyBullet

Data-driven methods in robot learning often require substantial amounts of data, which can be expensive to obtain in the real-world [96]. While simulators have been used in robotics to help alleviate this difficulty, such as for locomotion [202] and rigid object manipulation [152], many simulated benchmarks for manipulation focus on rigid objects, partially because of difficulties in simulation of deformables [8].

Motivated by these challenges, we provide support for deformable objects (called "soft bodies") in PyBullet [30], a widely-used publicly available simulator for robotics research. While prior work with soft bodies in PyBullet [41, 40, 139] use position-based dynamics solvers, we use new soft body physics simulation based on the Finite Element Method [12] with mass-springs and self-collisions among vertices [11]. Contact and friction constraints between soft bodies and multi bodies are solved in a unified constraint solver at the velocity level. Implicit damping of the velocity uses a Krylov style method and soft body contact and friction is based on Conjugate Gradient for symmetric positive definite systems and Conjugate Residual for indefinite systems [215].

In simulation, cables consist of a sequence of rigid bodies ("beads"). Fabrics and bags are soft bodies, where each consists of a set of vertices. For fabrics and bags, we create a plane and a sphere object, respectively, using Blender [28]. (For bags, we remove vertices to form an opening.) We then import the object mesh files into PyBullet. To improve physics stability of drawstring-style bagging behavior in PyBullet simulation, we additionally attach a ring of rigid beads at the bag's opening, using the same beads that form cables. See Figure 5.4 for an overview. We do not simulate thickness to these soft bodies, which can help with collision penetration, and which we leave for future work.

5.4.2 Benchmark for Manipulating Deformable Objects

We design 12 tasks with deformables, listed in Table 5.1. For consistency, each uses a standardized setup: a UR5 arm, a 0.5×1m tabletop workspace, and 3 calibrated RGB-D cameras diagonally overlooking the workspace, producing top-down observations $\mathbf{o}_t \in \mathbb{R}^{320\times160\times6}$ of pixel resolution 320×160, where each pixel represents a 3.125×3.125mm vertical column of 3D space. Observations contain 3 RGB channels and 3 channel-wise depth values. The goal is to train a policy that executes a sequence of pick and place actions in the workspace to achieve an objective, learned from demonstrations with behavior cloning [163]. Some tasks (*e.g., cable-shape-notarget* and *fabric-flat-notarget*) are specified by goal images of a target scene configuration, while others are described solely by the distribution of demonstrations (*e.g., cable-ring-notarget* and *bag-alone-open*).

The grasping motion primitive (introduced in Section 5.2) is similar to the implementations in [85, 184] and approximates a pinch-grasp on a deformable by indexing the nearest vertex that comes in contact with the end effector tip, and locking in its degrees of freedom to match that of the end effector using fixed constraints. Certain tasks are divided into different stages, such as *bag-items-1* and *bag-items-2* which involve opening a bag, inserting

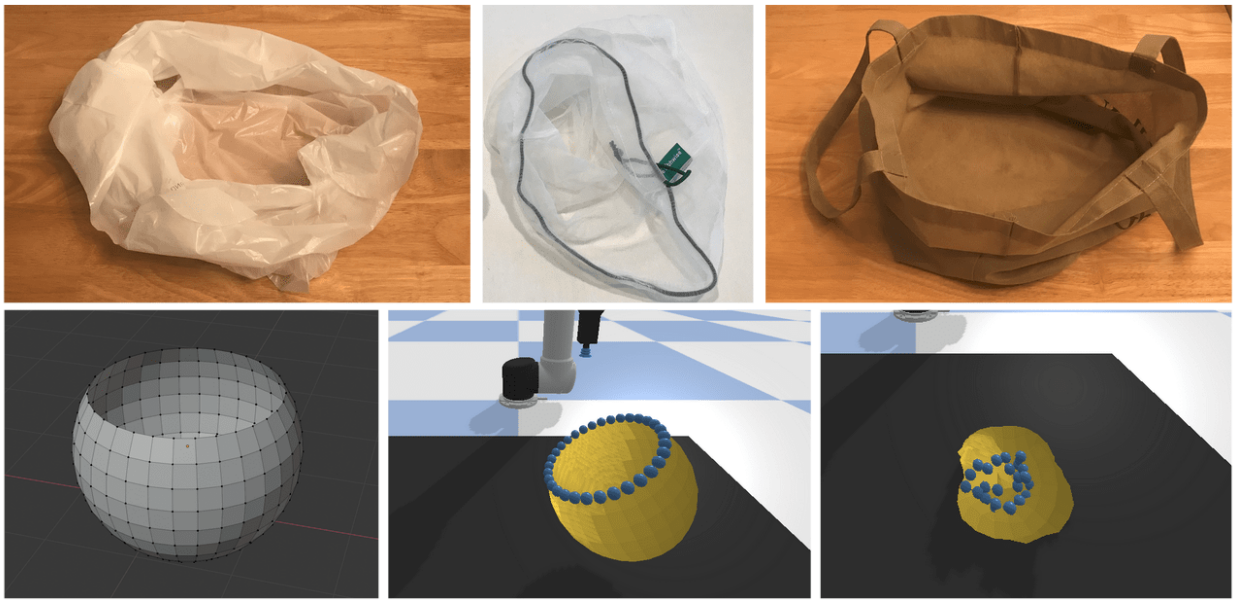

Figure 5.4: **Top row**: examples of physical bags. The bags we use follow a design similar to the sack (top left) and drawstring (top middle). Bags with handles (*e.g.,* top right) or with more stiffness will be addressed in future work. **Bottom row**: to make bags, we create a sphere in Blender and remove vertices above a certain height (bottom left). We import the mesh into PyBullet and add a series of beads at the bag opening. For bag tasks, we initialize the bag by randomly sampling a starting pose (bottom middle), applying a small force, and allowing the bag to crumple (bottom right).

items inside, and lifting a bag. Depending on the task stage and the gripped item, actions lift the gripped object to a different hard-coded height. For example, pulling the bag upwards requires a higher vertical displacement than opening the bag.

5.5 Experiments

We use scripted, stochastic demonstrator policies to get 1000 demonstrations (*i.e.,* episodes) per task, and train policies using 1, 10, 100, or all 1000 demonstrations. We provide a detailed overview of the data generation process in the Appendix. We test the following models:

- **Transporter.** This model is directly from [240].
- **Transporter-Goal-Split.** The Transport model that includes a separate goal module Φ_{goal} to process $\mathbf{o}_g$.

- **Transporter-Goal-Stack.** A model that stacks $\mathbf{o}_t$ and $\mathbf{o}_g$ channel-wise to form a 12-channel image, then passes it as input to standard Transporter Networks.
- **GT-State MLP.** A model that uses ground truth pose information as observations $\mathbf{o}_t$, and does not use images. It processes its input with a multi-layer perception (MLP) and directly regresses $\mathcal{T}_{\text{pick}}$ and $\mathcal{T}_{\text{place}}$.
- **GT-State MLP 2-Step.** A baseline similar to the prior ground truth baseline, except it regresses in a two-step fashion, first regressing $\mathcal{T}_{\text{pick}}$, and then concatenates this result again to the ground truth information to regress $\mathcal{T}_{\text{place}}$.

We test *Transporter* on non-goal conditioned tasks, whereas for those that use goals $\mathbf{o}_g$, we test with *Transporter-Goal-Split* and *Transporter-Goal-Stack*. Ground truth baselines are tested in both settings, where in the goal-conditioned case, we concatenate the ground truth pose information in both the current and goal configurations to form a single input vector. Following [240], to handle multi-modality of picking and placing distributions, the models output a mixture density [18] represented by a 26-D multivariate Gaussian. During training, all models use the same data augmentation procedures for a fair comparison.

Evaluation metrics Episodes for *cable-shape*, *cable-shape-notarget*, and *cable-line-notarget*, are successful if all cable beads have reached designated target poses. For *cable-ring*, *cable-ring-notarget*, and *bag-alone-open*, success is triggered when the convex hull area of the ring of beads (forming the cable or the bag opening) exceeds a threshold. Episodes for *fabric-cover* are successful if the fabric covers the cube. *Fabric-flat* and *fabric-flat-notarget* are evaluated based on coverage [184]. *Bag-items-1* and *bag-items-2* consider a success when all blocks have been transported to the target zone with the bag. Finally, *bag-color-goal* evaluates success if the item has been inserted into the correct colored bag.

5.6 Results

We summarize results in Table 5.2. For each model type and demo count $N \in \{1, 10, 100, 1000\}$, we train 3 models randomly initialized with different TensorFlow [138] seeds, except for *bag-color-goal*, where we report results from 1 model since the task is computationally expensive. Models are trained for 20K iterations, with a batch size of 1 for the three Transporter models and 128 for the two ground truth models (to strengthen the baseline). We save 10 snapshots throughout training at equally spaced intervals, and for each, we roll out 20 evaluation episodes. This gives 60 metrics (Section 5.5) for each snapshot, due to training 3 models. For each of the 10 snapshots and their associated metric, we average the 60 metrics and report the maximum over the 10 in Table 5.2. The Appendix has more extensive analysis.

Table 5.2: **Results.** Task success rate (mean % over 60 test-time episodes in simulation of the best saved snapshot) vs. # of demonstration episodes (1, 10, 100, or 1000) used in training. For the first eight tasks listed, we benchmark with Transporter Networks [240] ("Transporter") and two baselines that use ground-truth pose information instead of images as input. The last row of four tasks tests goal-conditioned policies, and we test with proposed goal-conditioned Transporter Network architectures (see Section 5.3.1), along with similar ground-truth baselines. We do not test the last four tasks with "Transporter" since the architecture does not support including an extra goal image as input. Section 5.5 and the Appendix contain more details.

	cable-ring				cable-ring-notarget				cable-shape				fabric-cover			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
GT-State MLP	0.0	0.0	0.0	0.0	0.0	1.7	3.3	5.0	0.4	0.8	1.0	0.5	3.3	25.0	18.3	21.7
GT-State MLP 2-Step	0.0	1.7	1.7	0.0	1.7	0.0	0.0	1.7	0.7	0.6	0.9	0.5	3.3	16.7	6.7	3.3
Transporter	16.7	50.0	55.0	68.3	15.0	68.3	73.3	70.0	75.6	80.6	90.1	86.5	85.0	100.0	100.0	100.0
	fabric-flat				bag-alone-open				bag-items-1				bag-items-2			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
GT-State MLP	26.0	45.6	65.6	71.3	15.0	16.7	35.0	43.3	1.7	20.0	30.0	31.7	0.0	0.0	6.7	8.3
GT-State MLP 2-Step	21.8	30.9	45.5	41.7	11.7	15.0	18.3	26.7	0.0	8.3	28.3	31.7	0.0	1.7	6.7	11.7
Transporter	42.1	86.5	89.5	88.8	18.3	50.0	61.7	63.3	25.0	36.7	48.3	51.7	5.0	30.0	41.7	46.7
	cable-line-notarget				cable-shape-notarget				fabric-flat-notarget				bag-color-goal			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
GT-State MLP	11.1	44.5	72.7	77.4	11.1	42.7	66.0	65.4	14.8	49.8	62.1	63.2	0.8	0.8	10.0	14.9
GT-State MLP 2-Step	8.5	39.4	58.8	65.4	9.4	44.9	54.9	56.4	23.0	51.1	59.6	61.9	4.9	5.0	5.0	15.0
Transporter-Goal-Stack	63.5	82.8	53.0	54.4	54.4	47.5	45.1	45.6	16.5	26.3	25.3	20.1	12.4	21.6	65.4*	70.3*
Transporter-Goal-Split	74.9	95.6	53.9	99.2	48.4	75.1	64.9	76.4	27.6	35.6	30.1	77.0	10.0	63.1	40.1*	49.8*

*trained with 40K iterations.

5.6.1 Non-Goal Conditioned Tasks

In these tasks, *Transporter* generally achieves orders of magnitude better sample efficiency than ground truth models. It performs reliably on *fabric-cover* with 100% success rates with just 10 demos (compared to 25% or less for all ground truth models), and does well on *cable-shape* ($\geq$ 90.1%) and *fabric-flat* ($\geq$ 86.5%) given enough demos. On *bag-alone-open*, *Transporter* attains performance of 61.7% and 63.3% with 100 and 1000 demos, which is comparable to the scripted demonstrator performance of 60.2% (1000 successes out of 1661) before data filtering. Similarly, on *bag-items-1* and *bag-items-2*, the best raw performance numbers (with 1000 demos) are 51.7% and 46.7%, which exceed demonstrator performance of 41.7% and 32.5% (see Table 5.1).

Qualitatively, the learned policies may generalize to new starting configurations from the same data distribution. Figure 5.1 shows an example successful rollout on *bag-items-1*, where the robot opens the bag, inserts the cube, picks and pulls the bag (which encloses the cube in it), then deposits the bag to the target zone. The project website contains videos.

5.6.2 Goal Conditioned Tasks

For *cable-line-notarget* and *cable-shape-notarget* with 1 and 10 demos, *Transporter-Goal-Stack* and *Transporter-Goal-Split* outperform ground-truth baselines, but the performance gap narrows with larger datasets, perhaps because ground-truth models should eventually do well given sufficient data. Furthermore, we observe from training curves (see the supplementary material) that at around 16-20K training iterations, both goal-conditioned Transporter models with larger datasets (100 and 1000 demos) start to get significantly higher test-time performance across the goal-conditioned tasks (especially *bag-color-goal*). From inspecting test roll outs, we hypothesize that these models need to train longer to disambiguate $\mathbf{o}_t$ and $\mathbf{o}_g$ for the picking network f_{pick}.

5.7 Conclusion and Future Work

In this chapter, we present a new suite of tasks that evaluate end-to-end vision-based manipulation of deformable objects, spanning cables, fabrics, and bags. In addition, we propose goal-conditioned model architectures with Transporter Networks for rearranging objects to match goal configurations.

Manipulating deformable objects is challenging, and while the proposed method yields promising results, its limitations point to interesting directions for future work. For example, Figure 5.5 shows several failure modes in the bag tasks. The bag opening step is challenging because a single counterproductive action that visually occludes the bag opening makes it difficult to recover. In addition, for *bag-items-1* and *bag-items-2*, items may fail to remain enclosed in the bag when lifted vertically. This is in part due to the discrete-time planar parameterization of our pick and place action space. Future work may investigate higher rates of control that learn recovery policies to react in real-time, or stateful systems that can

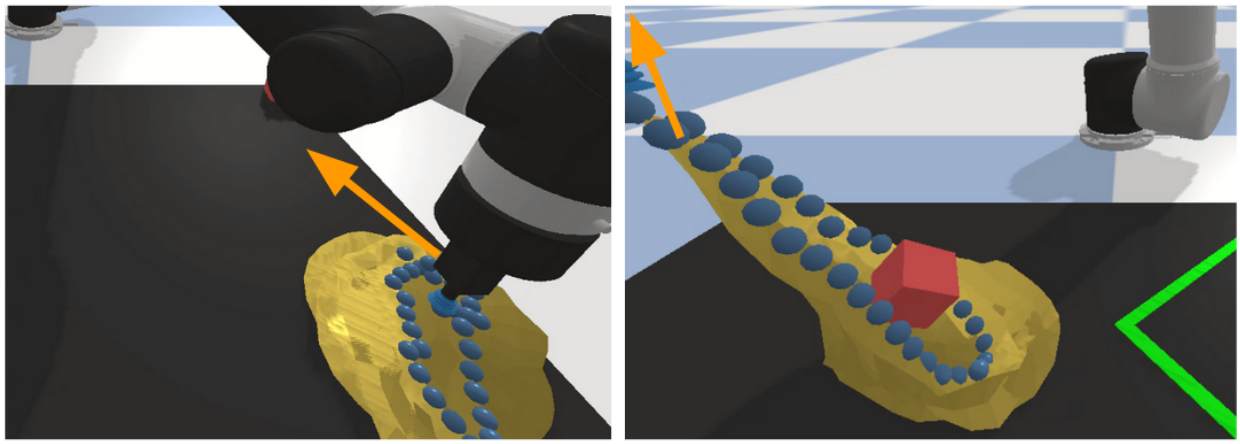

Figure 5.5: Failure cases we observe from trained Transporter policies on bag tasks. Left: in all bag tasks, a failure case may result from covering up the bag opening; these are hard for subsequent actions to recover from. Right: even if items are inserted into the bag, they may not be fully enclosed and can fall out when the bag is lifted. We overlay frames with orange arrows to indicate the direction of motion of the robot's end effector.

track objects under visual occlusions, or bimanual manipulation policies, which may help prevent items from falling out of bags. We also observe that the performance of policies on goal-conditioned tasks are slightly below those that explicitly rely on visual anchors in the workspace (*e.g.*, *fabric-flat* vs. *fabric-flat-notarget*), suggesting a performance gap. Finally, given the COVID-19 pandemic which limited access to robots, as conditions improve, we will conduct physical experiments with bags.

Part IV

Curriculum Learning

Chapter 6

ZPD Teaching Strategies for Deep Reinforcement Learning from Demonstrations

When a child follows an adult teaching by example, we want the child to gradually develop skills without assistance. Intuitively, an effective teaching method might provide examples that are within a "comfort zone" for the student — just difficult enough to learn from, but not so difficult that the student cannot benefit from the teaching. This intuition is formalized in "Zone of Proximal Development (ZPD)" theory, an epistemological theory built on a human cognitive model first proposed by social psychologist Lev Vygotsky [24, 216]. According to ZPD theory, learning examples are placed on a spectrum with problems that a learner can solve on their own on one side, and problems a learner cannot solve on the other. According to Vygotsky, when presenting a student with examples, those examples must be "in the Zone of Proximal Development," a set of problems which are difficult for the learner to solve alone, but which can be solved with guidance.

In this chapter, we investigate the application of ZPD in deep reinforcement learning contexts where a learner agent has access to several *teachers*. We define a teacher as a set of experience (*i.e.,* data samples) from a training model in a particular time window of its training, rather than a fixed policy. See Section 6.2 for a more formal overview.

The primary question we explore is the following: which among a given set of teachers should the agent "select" to best improve learning? While there has been substantial work on integrating reinforcement learning with demonstrations (e.g., DQfD [81] and DDPGfD [213]), to our knowledge there is no work investigating which among a set of teachers should be selected for teaching. We use off-policy Deep Q-learning algorithms with experience replay [126] to act both as the underlying student algorithm and for supplying teacher data. We evaluate on nine Atari 2600 benchmarks [15]. Our contributions include:

1. A ZPD-based algorithm that allows for dynamic selection of teachers for a given student agent.

2. Results showing that choosing the appropriate teaching strategy, which may not always be a single teacher, improves sample efficiency and may improve final reward.

The code is available at `https://github.com/DanielTakeshi/dqn`.

6.1 Related Work

We focus on applying ZPD concepts in off-policy deep reinforcement learning algorithms with experience replay, which allows sharing of samples between learners and teachers. Prominent examples of (deep) off-policy algorithms include the seminal Deep Q-Network (DQN) [144] and its notable extensions, such as Double DQN [73] which we employ in this work. Other improvements are accounted for in algorithms such as Rainbow [80] which combined multiple algorithmic components and showed complementary benefits. For simplicity, we use Double DQN (DDQN) as the base algorithm, but the algorithm we propose is broadly applicable to any off-policy, experience replay based agent.

Learning with Demonstrations Despite various advances in deep reinforcement learning, sample efficiency and exploration remain known challenges. A common ingredient to improve sample efficiency is using a demonstrator (equivalently in this work, a teacher). DAgger [170] is a popular approach, but requires a teacher to be continually present to label student-generated states with actions. Arguably a more practical alternative is to assume a fixed batch of data from (potentially multiple) demonstrators, without requiring any further teacher interaction, and to see how much an agent can learn from the data. If the agent does not do any environment interaction, the setting is known as *batch reinforcement learning* [58, 3, 114].

We similarly focus on off-policy learning with demonstrations, but allow the agent to take environment steps in addition to using demonstrator samples for learning. A prominent example of this for discrete control is Deep Q-learning from Demonstrations (DQfD) [81], the most relevant prior work, which seeded a learner agent with a small batch of human demonstrator data and showed performance gains on Atari games. Follow-up work, including Ape-X DQfD [161, 87] and R2D3 [156] scaled the idea to handle a larger stream of samples and a wider variety of environments. In continuous control several algorithms have built upon the Deep Deterministic Policy Gradients (DDPG) [125] algorithm such as in [213, 148, 212] to effectively leverage demonstrations for sparse reward tasks. Schmitt et al. [181] apply a similar idea in the actor-critic setting by adding an auxiliary cross-entropy loss function from [174] to encourage learner policies to be closer to a teacher's policy, and additionally consider an ensemble of teachers specialized to different tasks. None of these prior works, though, primarily focuses on determining which among a *set* of teachers we should select for using samples.

Distillation and Curriculum Learning The problem setting we consider can more generally be viewed as that of using a neural network teacher to accelerate the training of a neural network student. One of the most straightforward methods is to train a student model so that its final layer matches that of the teacher network. This is known as *distillation* and has been applied in classification [83, 60] and reinforcement learning [174] to decrease the size of neural networks and to improve task generalization. The method we propose is distinct but complementary to distillation. We focus on teaching via samples in a shared replay environment rather than matching final neural network layers, but the method of selecting appropriate teachers could be used for distillation based approaches, forming a curriculum [16] over teachers.

The use of curricula in machine learning and robotics [39] involves intelligently arranging samples to accelerate learning, typically from "easy" to "hard." A curriculum can designate a time-dependent and skill-dependent distribution of tasks [140] or a curriculum of start [53] or goal [52, 243] states. Curricula have also been used to accelerate training agents defined as a mixture model of policies of increasing complexity [32]. We refer the reader to [151, 195] for surveys of curriculum learning. To our knowledge, this chapter presents one of the first works on designing a curriculum for learning from progressively more advanced teachers.

6.2 Methodology

In a setup mirroring DQfD [81], we train student agents using DDQN [73] as the base algorithm, where training is based on a mixture of self-generated on-policy data and off-policy teacher data. Unlike in DQfD, we update the distribution of teacher data according to a "teaching strategy" f_{select} which draws experience from different teachers depending on the performance of the student. This teacher experience is blended in a mini-batch with student experience according to a second function f_{blend}. This differs from the classical literature in that the teacher experience now comes from a dynamic distribution instead of a fixed distribution drawn from the best performing teacher agent. An outline of the method is given in Algorithm 1.

6.2.1 Background

We use the standard Markov Decision Process (MDP) formulation for reinforcement learning [201]. An MDP consists of a five-tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where at each time step t, the agent is at a state $s_t \in \mathcal{S}$ from the set of possible states. For Atari environments, the states are of dimensions $\mathbb{R}^{84\times84\times4}$, and represent gray-scale environment images, consisting of four frames. The agent takes a (discrete) action $a_t \in \mathcal{A}$ from the set of possible actions, which in the Atari environments are game dependent. The environment dynamics map the state-action pair into a successor state $s_{t+1} \sim P(\cdot|s_t, a_t)$, and the agent receives a scalar reward signal $r_t = R(s_t, a_t)$. The objective is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted return $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$ with $\gamma \in (0, 1]$.

Algorithm 1 The Student-Teacher Framework

Input: A set of data from teacher snapshots, $\{\mathcal{D}_{T_1}, \ldots, \mathcal{D}_{T_N}\}$ corresponding to the teacher's training run, student policy π_θ, student replay buffer $\mathcal{D}_\text{S}$, gradient update frequency u.
$T_\text{ZPD} := f_\text{select}(\pi_\theta)$
for steps $t \in \{1, 2, \ldots\}$ **do**
  Take an environment step using π_θ, store resulting sample in $\mathcal{D}_\text{S}$
  if $t \mod u = 0$ **then**
    $\mathcal{D}_\text{minibatch} := f_\text{blend}(\mathcal{D}_{T_\text{ZPD}}, \mathcal{D}_\text{S})$
    Compute gradient update with $\mathcal{D}_\text{minibatch}$
    $T_\text{ZPD} := f_\text{select}(\pi_\theta)$
  end if
end for

6.2.2 The Student and Teacher

We consider the RL framework with *student* S, and *teacher* T, and denote policies as π_ψ with parameters ψ, which could represent either the student or the policy from the learning procedure that serves as the source for teacher data.[1] In our experiments we use the same neural network architecture for S and T, but this is not an assumption of the method, which treats the teacher as a black-box source of data.

In this chapter we use a set of contiguous time windows of samples from a single standard DDQN [73] training run to serve as teachers. The training uses the following loss function J_T for a given sample (s, a, r, s') to adjust its Q-network Q:

$$J_T(Q) = \Big(R(s,a) + \gamma Q(s', a'_\text{max}; \psi') - Q(s, a, ; \psi')\Big)^2 \tag{6.1}$$

where $a'_\text{max} = \arg\max_{a \in \mathcal{A}} Q(s', a; \psi)$, and ψ and ψ' are parameters of the current and target networks, respectively.

In the proposed algorithm, S can draw samples from both self-generated environment data, and from the teacher T. We train the student with a modified DDQN loss to account for the off-policy teacher samples which are injected into the student's replay buffer. The modified loss is the sum:

$$J_S(Q) = J_T(Q) + \lambda_E J_E(Q) + \lambda_2 J_{L_2}(Q) \tag{6.2}$$

where $J_E(Q)$ is the additional large margin classification loss [160], as used by DQfD [81]:

$$J_E(Q) = \max_{a \in \mathcal{A}} \Big[Q(s,a) + \ell(a_E, a)\Big] - Q(s, a_E) \tag{6.3}$$

[1] We use the terminology of "source for teacher data" rather than "teacher policy." This may reduce confusion, because we technically use different (contiguous) time windows of samples from the same learning model's training history as different "teachers" in our experiments.

and $J_{L_2}(Q)$ is an L_2 weight regularization of the Q-network. The λ_E and λ_2 are hyper-parameters controlling a trade-off between these loss values.

While we have designed the DDQN agents to closely model the original DQfD paper [81], our implementation differs in two major ways:

- We do not use the pre-training phase from DQfD in order to improve the generality of our results, and to explore training agents *de novo*.
- We do not use prioritization of the replay buffer [179] in any form in order to closely examine the effect of having a fixed minibatch sampling ratio (discussed in Section 6.2.4) on the performance of the model.

While it is possible to use an ensemble of independently trained agents as teachers, we instead draw our teachers from a set of fixed windows of samples generated during the training of a single DDQN agent. During the training process, we store snapshots of the parameters at regular time intervals with respect to environment steps. This results in a set of N saved snapshots $\{T_1, \ldots, T_N\}$ from the training history of T, each corresponding to a different policy. For the k-th teacher policy π_{T_k} from the training history, we denote a dataset of m states as

$$\mathcal{D}_{T_k} = \{s_0^k, s_1^k, \ldots, s_m^k\} \tag{6.4}$$

where, to make results more reproducible and to facilitate comparisons among different teaching methods, we take $\mathcal{D}_{T_k}$ to be a set of samples that the teacher agent experienced in a window centered around the time that the snapshot was saved. (We do not roll out π_{T_k} to get teacher data.) Thus, "teachers" are technically fixed contiguous windows of samples, and in an abuse of terminology, we sometimes use the term "teacher snapshot" to refer to these datasets.

6.2.3 Student/Teacher Matching with f_{select}

The ZPD teacher selection function f_{select} is a critical feature of Algorithm 1. This function determines the distribution across teachers from which the student samples experience. In our experiments, we draw only from a delta distribution across the teachers, and refer to the selected teacher as the *ZPD* teacher T_{ZPD}. We leave exploring prioritized sampling across multiple teachers to future work.

We propose defining f_{select} as a "fixed snapshots ahead" strategy, with an integer hyper-parameter k. For any student S, that student is matched to a snapshot T_{match}, the teacher with the reward closest to the one that the student achieves. We refer to the snapshot T_{match} as the "matched" teacher snapshot. The f_{select} function sets the ZPD teacher (the snapshot from which experience is drawn) to be snapshot $T_{\text{match}+k}$, capped at the last snapshot if it exceeds the total number of snapshots. Intuitively, the currently matched snapshot tells us the teacher snapshot that is roughly on par with the student, and so the next ZPD snapshot may be some fixed interval ahead, with higher rewards indicating that we are using teachers

that are more skilled at a given task. Another interpretation of this matching function is as a curriculum [16] where the learner initially starts with a teacher that provides "easier" samples, before moving on to harder teachers.

While it is possible that updating the ZPD teacher at each time-step would provide the best performance, for efficiency, we update the ZPD teacher infrequently. A call to f_{select} triggers when the student's average reward (over a window of 100 episodes) exceeds that of the matched teacher's reward, the latter of which is the average over a window of 100 episodes the teacher experienced at the time the snapshot was saved.

In this work, we examine fixed snapshot ahead strategies for $k \in \{0, 2, 5, 10\}$ and call these "k ahead" methods. We additionally test with two baselines: a "best ahead" where the ZPD snapshot is fixed throughout student training to be the teacher snapshot with highest reward, and a "random ahead" where the next ZPD snapshot is chosen at random among the set of all teacher snapshots with higher reward than the student.

6.2.4 Minibatch Sampling with f_{blend}

Once a ZPD teacher, T_{ZPD}, is selected, we can treat the corresponding dataset $\mathcal{D}_{T_{\text{ZPD}}}$ of samples as a distinct "ZPD teacher" replay buffer. We then sample from both $\mathcal{D}_{\text{S}}$ and $\mathcal{D}_{T_{\text{ZPD}}}$ in each minibatch update of the DQfD algorithm. When sampling, however, we need to decide how many samples to draw from the teacher's replay buffer (off-policy examples) and how many to draw from the replay buffer of the student (on-policy examples). This trade-off is made according to the strategy f_{blend}, a binomial distribution representing the probability that a sample in the minibatch is drawn from a particular replay buffer.

In this work, we use a fixed ratio of student-teacher samples per mini-batch. While prior work has tested using a ratio of 25% teacher samples in a minibatch as in Ape-X DQfD [161] or even smaller ratios of 1/256 as in R2D3 [156], we use a 50% ratio to test the effect of using more off-policy teacher data per minibatch. These ratios are held constant throughout student training, up to the point when the student surpasses every teacher snapshot. When this occurs, we anneal f_{blend}, in a linear decay towards no teacher samples. At this point, the student has little to learn from the teachers.

6.3 Experimental Setup

We evaluate our methods using the Atari [15] benchmarks for the following nine games: Alien, BeamRider, Boxing, Breakout, Pong, Qbert, Robotank, Seaquest, and SpaceInvaders. These include several widely-used games (Pong and Breakout in particular) and others for which DDQN can get reasonable performance. We run into a limitation for complex games with sparse rewards, such as Montezuma's Revenge and Private Eye. For those games, training a teacher agent via deep reinforcement learning is famously difficult, and successful learning normally requires some combination of human demonstrators, the underlying game state, or advanced exploration strategies with massively parallel data collection [176, 23].

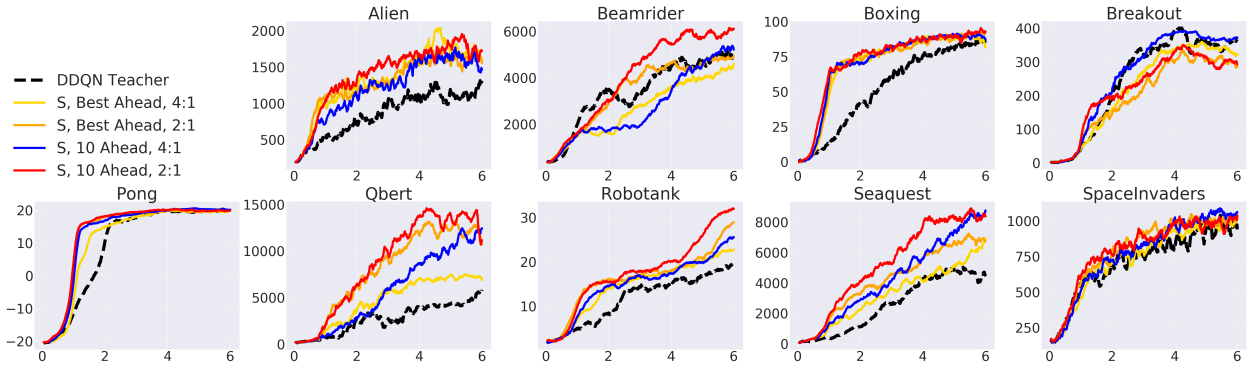

Figure 6.1: Learning curves for all nine games tested, comparing: 10 ahead, best ahead, and the original DDQN teacher. For 10 ahead and best ahead, we test with both a 4:1 and a 2:1 ratio of steps:updates, as discussed in Section 2.2.1. Learning curves are over two random seeds. Results suggest that the 2:1 throughput case is at or exceeds the level of the 4:1 case across games.

Table 6.1: Hyperparameters used when training teachers and then students. In addition to values listed here, both the student and teachers used an Adam [101] learning rate of 10^{-4}, batch size 32, target network sync-ing period of 10,000 steps, and discount factor $\gamma = 0.99$.

Hyperparameter	**Teacher**	**Student**
Environment steps	6,000,000	6,000,000
Replay buffer size	1,000,000	250,000
Steps between gradient updates	4	2
Student-teacher minibatch ratio	N/A	50-50
L_2 regularization weight λ_{L_2}	N/A	10^{-5}
Supervised loss weight λ_E	N/A	10^{-1} or 10^{-2}
Supervised loss margin ℓ	N/A	0.8
Size of ZPD teacher buffer $\mathcal{D}_{T_{\text{ZPD}}}$	N/A	≈ 250000

Since we compare performance of students versus teachers, we wish to have teachers follow the same base algorithm of DDQN, so we did not test on games for which DDQN performs extremely poorly.

For all teacher DDQN training runs, we saved 23 snapshots, spaced out over 250,000 environment training steps out of the 6 million total steps taken, where each environment step is exactly 4 frames, following OpenAI gym [22] conventions. The first teacher snapshot was saved at 400,000 steps, the second at 750,000 steps, and continued until the 23rd at 5,900,000 steps. We use 30 random no-op actions to begin each environment. For each teacher selection method, we benchmark with two random seeds and average the results.

We test the following six f_{select} methods: 0 ahead, 2 ahead, 5 ahead, 10 ahead, best ahead, and random ahead (discussed in Section 6.2.3). For the DQfD lambda hyperparameter λ_E,

which controls the strength of the imitation loss, we tuned values in $\{0, 0.001, 0.01, 0.1, 1.0\}$ for all nine games with the 0.8 large margin loss from [81]. In contrast to their work, we found that $\lambda_E = 1.0$ was too large and resulted in deficient student performance, perhaps because we use relatively more off-policy data. Based on initial tuning, six games worked best with $\lambda = 0.1$ and the other three (BeamRider, Pong, Robotank) worked best with $\lambda = 0.01$. Our main hyperparameters are shown in Table 6.1, where we borrow some relevant hyperparameters from DQfD [81] to minimize tuning, and reduce the size of the experience replay buffer [242].

We also tuned the number of environment steps taken by the student in between gradient updates. The standard in DQN is to use 4 environment steps for every gradient update. With 50% of the minibatch coming from demonstrator samples, doing a 4:1 step:update ratio means that each new student environment step sample is consumed, in expectation, half as many times as in the no-teacher case. Using a 2:1 ratio, for the same number of environment steps, means the student can perform twice as many gradient updates, which may improve learning. We thus tested both 4:1 and 2:1 ratios across all games and teaching methods. Figure 6.1 shows a representative set of results, suggesting that students trained with a 2:1 ratio outperform those with a 4:1 ratio. Thus, for all results we use a 2:1 ratio, and leave examining the step-to-update ratio for "student throughput" to future work.

6.4 Results

Table 6.2: A ranking of the various methods across the nine games, combining all trials from Figure 6.2. We indicate the number of times the method "won", i.e., had highest reward (over two random seeds) for a given game.

Method	BA	RA	0A	2A	5A	10A
Average	0	2	0	0	3	4
Last 100	1	3	1	0	3	1

Figure 6.2 plots the learning curves, averaged over two random seeds, for all nine games and all teaching strategies tested. Traditional methods such as DQfD rely on the best ahead style of teaching — they choose a constant teacher achieving the highest final reward. We can see that by dynamically altering the teaching strategy, in almost all cases the best ahead strategy is sub-optimal with respect to both final reward and sample efficiency. Table 6.2 summarizes this observation.

The learning curves and tables suggest that the strongest teaching methods are the random ahead, 5 ahead, and 10 ahead methods. This result not only supports our overall hypothesis that a dynamic ZPD style strategy can improve performance, but also indicates that it may not be necessary to use the very highest-rewarding expert available in all circumstances. One possible explanation for these results is that with high-quality samples

from the teacher with highest reward, running a DQfD-style algorithm may overfit to that teacher's samples (as overfitting has been shown in deep Q-learning [56]). Also intriguing is the performance of the random ahead method. The diversity of samples introduced by the method could aid in avoiding overfitting to a given teacher or a given skill range. We believe that it is interesting future work to further explore these random teaching methods, as additional exploration could yield insights into the learning process of Deep Q Networks.

Another clear result is that, as expected, the best teaching methods for a particular game outperform the teacher itself. This is despite the 50-50 minibatch ratio, which uses a relatively large amount of off-policy data as compared to DQfD [81], Ape-X DQfD [161], and especially R2D3 [156], which complicates learning as Q-learning algorithms normally need significant on-policy experience [58].

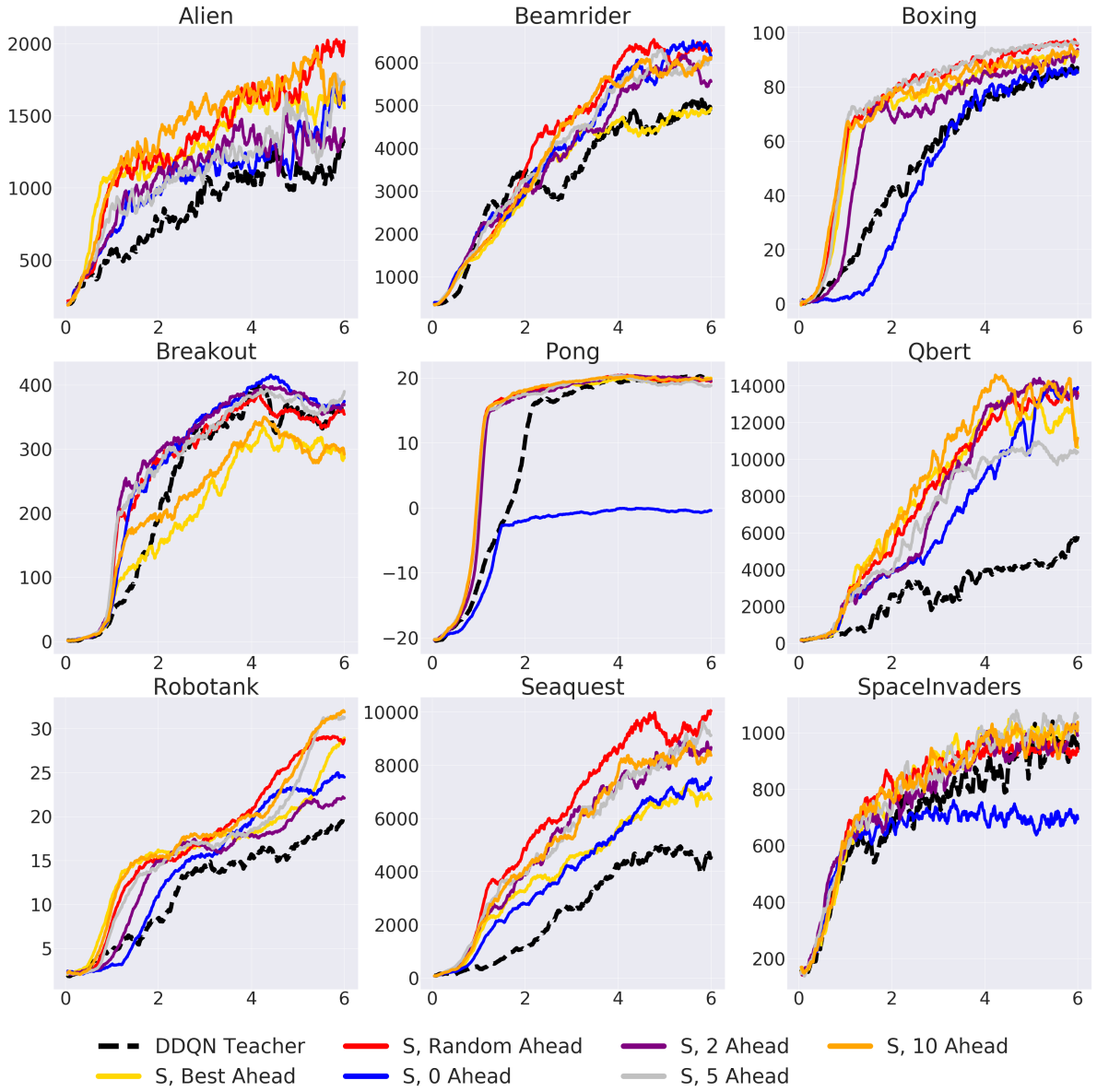

Figure 6.2: Results for f_{select} methods as described in Section 6.2.3 on the nine Atari games we use. Learning curves represent the average reward over a window of 100 past episodes as a function of environment steps (six million total) as suggested in [132]. Curves are then re-averaged over two random seeds per f_{select} function. Each seed for a game is run with the *same* teacher and the *same* set of datasets generated from that teacher to reduce the variability in results. The black dashed lines represent the performance of the DDQN teacher agent, run only once per game. Results suggest that using the best ahead matching, indicated in bright gold above, is not the clear best choice for any of the games tested.

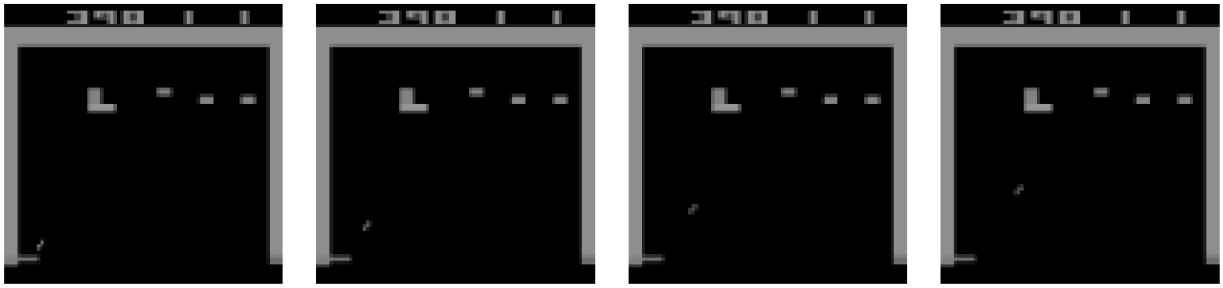

Figure 6.3: A representative example of a set of Breakout frames that an expert agent frequently encounters. Here, the agent moves back and forth repeatedly in the same pattern and the ball does not hit any remaining bricks. The game only exits this "deadlock" due to either the 1% chance of a random action from the epsilon greedy policy or the 10K episode step limit.

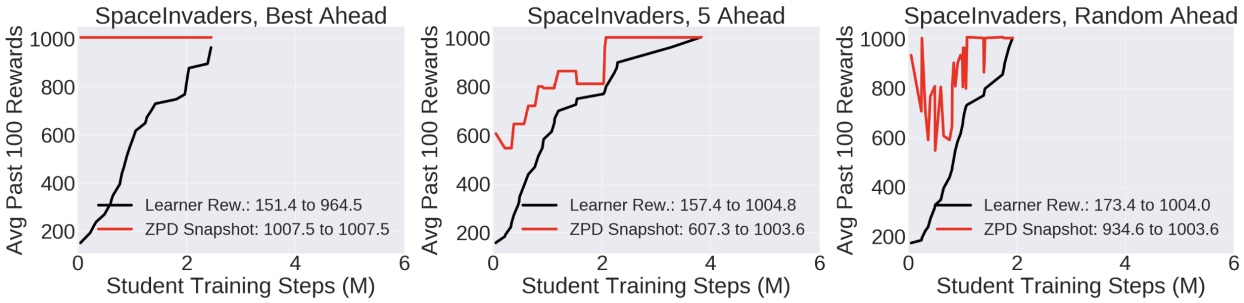

Figure 6.4: Examples of learner reward curves and corresponding ZPD reward values for Space Invaders for f_{select} functions: best ahead (left), 5 ahead (middle), and random ahead (right). The plots only show the curves for the period when the learner has lower reward than its ZPD teacher. At an environment step, the reward of the learner and the reward of its ZPD teacher is plotted, showing that the best ahead method always uses a ZPD teacher with the highest possible reward, while the random method uses ZPD teachers with varying reward levels (that exceed the reward of the learner).

6.4.1 Per-Game Analysis

The results indicate that the best strategy for making use of teacher samples appears to be game-dependent. From Table 6.2, the best ahead method is only highest in one case, Pong, when considering the average of the last 100 episodes. The learning curves indicate Pong is unique: all teaching methods, with the exception of 0 ahead, have near identical performance. This phenomenon is surprising when considering that, for example, the 2 ahead method will initially query samples from a ZPD teacher that attains roughly -17 reward. Episodic rewards for Pong range from -21 to 21, so this suggests that using samples

from teachers with low game rewards can nonetheless be beneficial.

Breakout is another interesting case study. The learning curves suggest that the best ahead and the 10 ahead strategies are the two weakest strategies for the learner. One hypothesis for why Breakout may be better suited to a shorter ZPD "advancement interval" is because more advanced agents are frequently in states that might not be beneficial to a learner. Upon observing trained teacher DDQN policies, we noticed that they often got stuck in an endless cycle of hitting the ball back and forth in the same pattern in an effort to get the last few remaining bricks. Figure 6.3 shows a representative example from Breakout. Advanced agents are frequently in complex states, which intuitively might not be as helpful to beginning learners. Thus, for faster learning, it may initially make more sense to use samples from a a slightly weaker agent.

Other interesting results come from Robotank and Seaquest. While results suggest that using any of the selection functions results in improvements over a teacher (which may in large part be due to more frequent gradient updates), there exists a wide range in performance among the selection methods. Thus, careful tuning may be beneficial, and it would be interesting to see the effects of teaching methods when training for more environment steps.

Figure 6.4 provides some intuition on the matching process for various f_{select} methods. The figure shows, for the same Space Invaders teacher, reward curves for the student under a representative best ahead, 5 ahead, and random ahead training run. We can see that the best ahead method uses the same ZPD teacher with reward 1007.5. The 5 ahead method starts with a ZPD teacher at reward 600 and then gradually the reward increases to the reward of the final snapshot. The ZPD reward is not monotonically increasing because it is not generally the case that reward always increases throughout the 23 saved snapshots.

It is interesting to see in Figure 6.4 that the random ahead strategy experiences a more diverse reward range from its ZPD teachers. We believe that investigating this phenomenon in future work is important; can we understand and quantify the benefit one gets from randomizing a teacher?

6.5 Conclusion and Future Work

In this chapter, we proposed an approach that merges the Zone of Proximal Development (ZPD) theory from psychology with modern deep reinforcement learning practices. We demonstrated an algorithm for determining which of several teachers should be selected to provide samples for a learner, and demonstrated results on Atari environments suggesting that it is not ideal to solely rely on the highest-reward teacher.

We also raise a number of interesting questions for consideration in future work. For example, can we choose a dynamic f_{select} function which does more than take a fixed number of k steps ahead? One possibility would be defining a function which relies on the overlap of the state distribution of the teacher and learner. We would also like to explore how the proposed method may be used in combination with ideas from batch-constrained learning [58], or Random Ensemble Mixtures [3] to better train agents entirely off-policy.

While it is only a single step towards developing complex teaching strategies, this chapter shows that defining teaching strategies should serve as an important step in the deep learning from demonstration pipeline, and that by choosing naive methods we are ignoring significant possible performance gains.

Chapter 7

DCUR: Data Curriculum for Teaching via Samples with Reinforcement Learning

Humans often learn best when guided through a curricula. When providing expert demonstrations to human students, the demonstrations should fall within a particular range of difficulties. If they are too easy the student learns nothing, but if they are too difficult the student may have trouble learning [24, 216]. With this intuition, we consider the analogous context in Reinforcement Learning (RL) [201], and study how a teacher policy can best "teach" a student policy, where both are trained with reinforcement learning. In this work, we investigate when a teacher provides the student with a dataset of samples (*i.e.,* data tuples) $\mathcal{D}^{(T)}$. For the student, rather than propose a new algorithm to learn from data, we use *existing* algorithms but focus on *data usage*. In particular, given a student employing a standard off-the-shelf RL algorithm, how can it better sample from $\mathcal{D}^{(T)}$?

We propose a novel framework, **DCUR** (pronounced *dee-curr*): **D**ata **CU**rriculum for **R**einforcement learning, where we study how to filter a teacher's static dataset to accelerate a student's learning progress, measured in terms of environment episodic reward. The framework is compatible with a variety of student learning settings, including pure offline reinforcement learning [43, 116, 121] and scenarios where the student can engage in a small amount of self-generated data, which we refer to as apprenticeship learning. Either case reduces the need for the student to engage in extensive and potentially risky exploratory behavior, and thus may hold tremendous promise for enabling robots to learn from existing, massive datasets. In experiments, we test the DCUR framework by training teachers in a standard fashion with online environment interaction. We store its logged history of experienced environment interactions as a large dataset $\mathcal{D}^{(T)}$ to be used by the student for learning. Results over a range of standard continuous control tasks [22, 208] suggest that students running off-the-shelf, off-policy RL algorithms can make use of curricula to more efficiently learn from static teacher data, even without the use of specialized offline RL algorithms. See Figure 7.1 for an overview of DCUR. In summary, the contributions of this

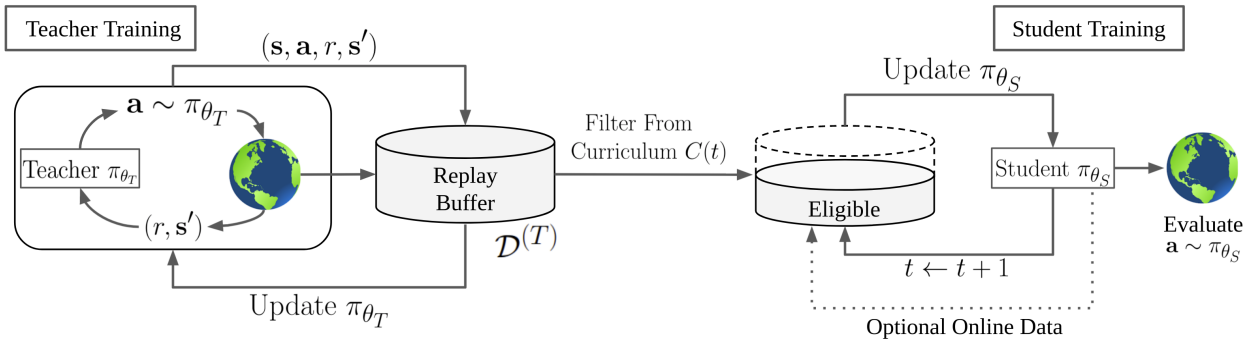

Figure 7.1: Visualization of DCUR. First, a *teacher* policy π_{θ_T} is trained using standard Deep RL with online environment interactions, and fills a replay buffer $\mathcal{D}^{(T)}$ with all the experienced data tuples. After training π_{θ_T}, we choose a fixed data curriculum $C(t)$ (see Figure 7.2) for training a *student* policy π_{θ_S} from $\mathcal{D}^{(T)}$. In general, we study when students train *offline*, but since this can be challenging, we optionally allow students to gather some online data to use for learning, in addition to $\mathcal{D}^{(T)}$. For each time t (*i.e.,* minibatch gradient update), the fixed curriculum strategy $C(t)$ restricts the samples the student can draw from $\mathcal{D}^{(T)}$, resulting in a set of eligible data tuples (shaded in light gray) that the student can use for gradient updates.

chapter include:

- We introduce the DCUR framework which considers data improvements, rather than algorithmic improvements, for accelerating learning from large, teacher-provided datasets.
- We show that the best curricula, combined with potentially longer training, can enable students training offline to match the teacher's top performance.
- We show that students can use some online data (2.5-5.0% of the offline data size) in combination with data curricula to aid learning in more complex environments.

Supplementary material is available at `https://sites.google.com/view/teach-curr/home`.

7.1 Related Work

This work builds upon an extensive literature in subfields including offline reinforcement learning, learning from demonstrations, and curriculum learning.

Curriculum Learning. The use of curriculum learning in machine learning dates to at least Elman et al. [39], who showed the benefit of initially training on "easier" samples while gradually increasing the difficulty. Subsequent work by Bengio et al. [16] confirmed

these results by accelerating classification and language modeling by arranging samples in order of difficulty. Other work in curriculum learning has included training teacher agents to provide samples or loss functions to a student [46, 224], and larger-scale studies to investigate curricula for image classification [225]. In RL, curriculum learning has shown promise in multi-task [140, 92] and self-play [196] contexts, for selecting one of several teachers to provide samples [188], and for generating a curriculum of start [53] or goal [52, 243] states. In this work, we design a curriculum of samples (*i.e.,* data tuples) for (mostly) offline RL, and we do not focus on the multi-task setting nor do we require self-play or goal generation. When students execute online steps, this can be interpreted as a form of apprenticeship learning [1] where the student can "practice" in addition to using offline data.

Offline Reinforcement Learning. Offline RL [121], also referred to as Batch RL [43, 116], has seen an explosion of recent interest. Offline RL is the special case of reinforcement learning [201] without exploration, so the agent must learn from a static dataset. It differs from imitation learning [154] in that data is annotated with rewards, which can be utilized by reinforcement learning algorithms to learn better policies than the underlying data generating policy. Many widely utilized Deep RL algorithms, such as DQN [144, 73] for discrete control and DDPG [125], SAC [69], and TD3 [57] for continuous control are off-policy algorithms and, in principle, capable of learning offline. In practice, however, researchers have found that such off-policy algorithms are highly susceptible to bootstrapping errors and thus may diverge quickly and perform poorly [58, 59, 111, 114, 113]. One remedy is to incorporate conservatism in reinforcement learning such as by regularizing the value functions in model-free RL settings [58, 114, 112, 244, 193, 226, 219]. Other studies have found promising results in model-based contexts [233, 99, 232]. The goals of this work are orthogonal to work that attempts to develop specialized offline RL algorithms, because the focus here is on knowledge transfer from a teacher to a student, with the offline setting as one possible learning scenario for the student.

Reinforcement Learning with Demonstrations. Combining reinforcement learning with demonstrations is a highly effective technique for training students, particularly for hard exploration environments. In off-policy RL, demonstrations can be inserted into a replay buffer [126] and sampled for learning along with self-generated samples from a student. Examples of such algorithms in discrete control settings include DQfD [81], Ape-X DQfD [161], and R2D3 [156]. Other work has utilized demonstrations in continuous control by adding transitions to a replay buffer [148, 213, 212], specifying an auxiliary loss [181] or estimating value functions for model-based RL [206]. These works enable additional exploration from the student, allowing for self-generated samples, whereas we aim to understand how well students can learn with minimal exploration. Furthermore, these works often use a very low *demo ratio*, or the fraction of expert (teacher) demonstrations in a given minibatch. For example, R2D3 [156] reported that a demo ratio as small as 1/256 was ideal. That these

algorithms perform best when utilizing so little teacher data motivates the need to understand how to use teacher data without requiring frequent student environment interaction.

7.2 Problem Statement and Preliminaries

We utilize the Markov Decision Process (MDP) framework for reinforcement learning (RL) [201]. An MDP is specified as a tuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where at each time step t, the agent is at state $\mathbf{s}_t \in \mathcal{S}$ and executes action $\mathbf{a}_t \in \mathcal{A}$. The environment dynamics map the state-action pair into a successor state $\mathbf{s}_{t+1} \sim P(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$, and the agent receives a scalar reward signal $r_t = R(\mathbf{s}_t, \mathbf{a}_t)$. The objective is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected discounted return $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$ with discount $\gamma \in (0, 1]$. In Deep RL, the policy π_θ is parameterized by a deep neural network with parameters θ.

The RL framework we study involves two agents: a *student* S and a *teacher* T, following respective policies π_{θ_S} and π_{θ_T} with parameters θ_S and θ_T. We assume the teacher is trained via standard online RL, and produces a dataset $\mathcal{D}^{(T)} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}_{i=1}^{N}$ of N *data tuples* (also referred to as "samples"), where each contains a state $\mathbf{s}_i$, action $\mathbf{a}_i$, scalar reward r_i, and successor state $\mathbf{s}_{i+1}$. In general, we use the i subscript notation in $(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})$ to specify a time indexing of the tuples across the full data, and use $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ when precise time indexing is not needed. Data tuples from teacher data $\mathcal{D}^{(T)}$ are provided to the student S, which we assume runs an off-policy RL algorithm, so that it can in principle learn from just the fixed data. We consider the problem of designing a curriculum to decide, for each time step t of the student's learning progress,[1] which data tuples from $\mathcal{D}^{(T)}$ should be "available" to the student when it performs minibatch gradient updates.

7.3 Methodology

7.3.1 Teachers and Data Generation

The DCUR framework is agnostic to the precise algorithm to train students and teachers. Unless stated otherwise, we generate teacher data $\mathcal{D}^{(T)}$ using Twin-Delayed Deep Deterministic Policy Gradients (TD3) [57], a state-of-the-art off-policy Deep RL algorithm for continuous control. TD3 is an actor-critic algorithm where the actor π_{θ_T} is a policy, and the critic is a value function which consists of two Q-networks, Q_{ϕ_1} and Q_{ϕ_2}, with target networks $Q_{\phi_1,\text{targ}}, Q_{\phi_2,\text{targ}}$. During gradient updates, TD3 mitigates overestimation of Q-values by taking the minimum of the two Q-networks to compute targets y for the Bellman update:

$$y = r + \gamma \min_{i \in \{1,2\}} Q_{\phi_i,\text{targ}}(\mathbf{s}', \mathbf{a}'(\mathbf{s}')) \tag{7.1}$$

[1] In this work, we consider RL contexts where it is standard to have a 1:1 ratio of environment steps and gradient (*i.e.,* training) updates, modulo any initial online data collection to partially fill in a replay buffer before training begins. We thus treat a "time step" t as referring to environment steps and gradient updates interchangeably. If students learn offline, then a "time step" is interpreted as a gradient update only.

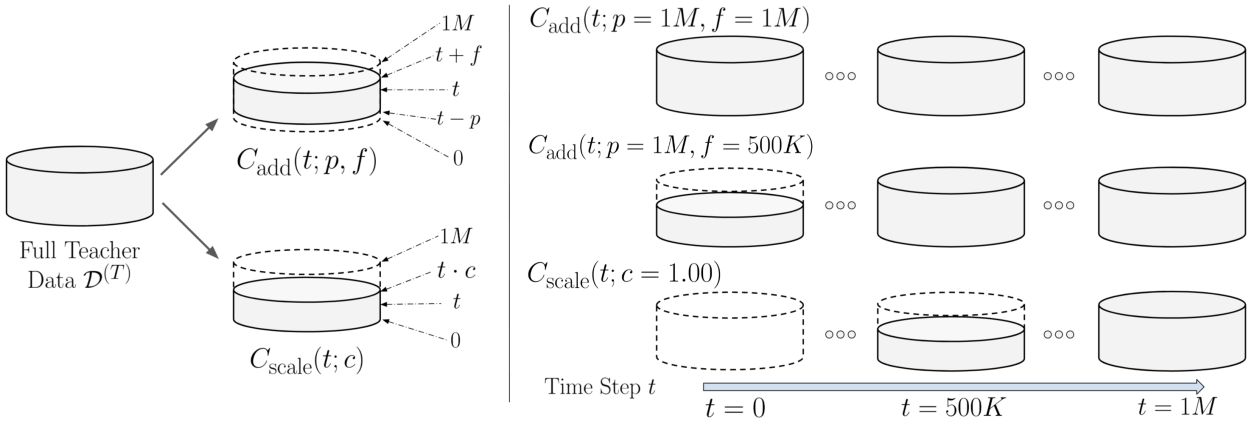

Figure 7.2: Visualization of the data curricula described in Section 7.3.2. For all buffer visuals, the bottom represents index 0, *i.e.,* the first data tuple from the teacher's training history, and the top is index 1M. At a student training step t, data tuples available for sampling are colored gray. **Left**: we illustrate the $C_{\text{add}}(t)$ and $C_{\text{scale}}(t)$ curricula, which determine different ranges of data tuples in $\mathcal{D}^{(T)}$. **Right**: three examples of curricula (one per row) showing how the available data to the student changes over 1M training time steps. For $C_{\text{add}}(t; p = 1M, f = 1M)$, the full $\mathcal{D}^{(T)}$ buffer is available at all times, whereas $C_{\text{scale}}(t; c = 1.00)$ only enables indices 0 to t for time t, and hence the available samples grows throughout training.

with discount factor γ, and where $\mathbf{a}'$ is the action considered at the successor state $\mathbf{s}'$:

$$\mathbf{a}'(\mathbf{s}') = \text{clip}(\pi_{\theta_T}(\mathbf{s}') + \epsilon, \mathbf{a}_{\text{low}}, \mathbf{a}_{\text{high}}), \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -\beta, \beta) \tag{7.2}$$

which in practice involves adding zero-mean clipped Gaussian noise ϵ to $\pi_{\theta_T}(\mathbf{s}')$ for some β, then clipping (a second time) component-wise to an environment-dependent action range $[\mathbf{a}_{\text{low}}, \mathbf{a}_{\text{high}}]$. For more details on TD3, we refer the reader to Fujimoto et al. [57]. To generate $\mathcal{D}^{(T)}$, we use the *logged environment interaction history* of TD3 from online training, resulting in a set of ordered tuples:

$$\mathcal{D}^{(T)} = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}_{i=0}^{N=1M}, \tag{7.3}$$

in a *replay buffer*, where following standard MuJoCo training [57], the number of environment steps and the buffer capacity are both 1M, so no data tuples are overwritten.

The students also use TD3 as the base algorithm. To our knowledge, the only prior work that has tested TD3 in offline RL with logged data is from Agarwal et al. [3], who report that TD3 outperformed Batch Constrained Q-learning [58] when learning from logged data generated from DDPG agents. We perform a deeper investigation of training on logged data by showing the utility of curricula (Section 7.3.2) and self-generated data (Section 7.3.3).

7.3.2 Data CUrriculum for Reinforcement Learning (DCUR) Framework

We propose to accelerate student learning (using TD3 as the base algorithm) with a curriculum $C(t)$ which specifies the range of eligible data tuples in the static teacher data $\mathcal{D}^{(T)}$ which can be sampled for the minibatch gradient update at time t. We define two classes of curricula: *additive* and *scale*. An additive curricula C_{add} uses two parameters, $p \geq 0$ and $f \geq 0$, specifying the *previous* and *future* data tuples in $\mathcal{D}^{(T)}$ *relative to* t. For ease of notation, we omit the p parameter if the intent is to always allow data tuples from index 0, which represents the teacher's earliest environment interaction. We thus denote the additive curricula on $\mathcal{D}^{(T)}$ using one of two conventions:

$$\begin{aligned} C_{\text{add}}(t; p, f) &= \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}_{i=\max(0,t-p)}^{i=\min(1M,t+f)} \\ C_{\text{add}}(t; f) &= \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}_{i=0}^{i=\min(1M,t+f)} \end{aligned} \tag{7.4}$$

where the valid time indices are centered at the current student training time cursor t, and are always limited by the buffer data size of $|\mathcal{D}^{(T)}| = 1M$ studied in this work. The second class of curricula C_{scale}, is parameterized by a single *scale* parameter $c > 0$ and defined as:

$$C_{\text{scale}}(t; c) = \{(\mathbf{s}_i, \mathbf{a}_i, r_i, \mathbf{s}_{i+1})\}_{i=0}^{i=\min(1M,t\cdot c)} \tag{7.5}$$

which enables index 0 up to index $t \cdot c$. See Figure 7.2 for a visualization. If the student trains for 1M gradient updates following $C_{\text{add}}(t; f = 1M)$, it has the full buffer accessible for sampling data tuples at all times. Using $C_{\text{scale}}(t; c = 1.00)$ means the student can only sample from indices 0 up to t at time t, so the available offline data grows over time. In addition, $C_{\text{add}}(t; f = 0)$ and $C_{\text{scale}}(t; c = 1.00)$ define the same data curriculum, and we default to using the latter notation.

To ground this with a concrete example, if the curriculum is set at $C_{\text{add}}(t; f = 50K)$, then at time $t = 10K$ near the start of the student's training, it can access samples in the teacher buffer from indices 0 up to 60K. At the halfway point of $t = 500K$ for student training, it has access to teacher samples from indices 0 to 550K, and then at $t = 950K$, it now has access to all of the teacher samples, which remains the same up until $t = 1M$.

In prior work, Fujimoto et al. [58] tested two special cases of these curricula, named *final buffer* and *concurrent*. The final buffer setting enables the entire data at all times for learning and was later studied in Agarwal et al. [3]. This is equivalent to $C_{\text{add}}(t; f = 1M)$. The concurrent setting is represented as $C_{\text{add}}(t; c = 1.00)$. We remark that that limiting the size of the buffer is a known option to stabilize online RL [242], but we aim to study this in an largely offline context in more complex environments, and where the eligible data buffer can grow over time.

7.3.3 Apprenticeship Learning: Using a Small Amount of Online Data

While we primarily study the student learning purely offline, we additionally explore learning with small amounts of on-policy student data, to see how much this stabilizes training and to also check that such effects are complementary with a data curriculum. This means the student forms a smaller buffer $\mathcal{D}^{(S)}$ of self-generated online data with exploration noise, and where $|\mathcal{D}^{(S)}| \ll |\mathcal{D}^{(T)}|$. We call this setting *X% Online* if the student, by the end of its training, has collected self-generated data tuples that amount to X% of the full teacher data (1M in this work). The student still performs 1M gradient updates, but takes one online step at equally spaced time intervals (*i.e.,* one step every 100/X updates). For example, with "5% Online," the student takes online environment steps 1 out of every 20 gradient updates, and $\mathcal{D}^{(S)}$ contains 50K data tuples at the end of training. This can equivalently be viewed as the student performing 50K consecutive online environment steps, but with 20 gradient updates between consecutive steps. The data tuples from $\mathcal{D}^{(S)}$ are never discarded. When the student samples a minibatch at time t, it first applies the pre-selected curriculum (see Section 7.3.2) to get the appropriate subset of $\mathcal{D}^{(T)}$, then combines the resulting eligible data tuples with all of $\mathcal{D}^{(S)}$, then uniformly samples from that.

7.4 Experiments

We provide an overview of the experiments and defer further details to the Appendix.

Environments We test DCUR using standard MuJoCo environments [208] for Deep RL: Ant-v3, HalfCheetah-v3, Hopper-v3, and Walker2d-v3 from OpenAI [22]. Earlier versions of these environments (-v1 or -v2) have also been benchmarked in work focusing on offline RL [58, 114, 226].

Teachers and Students We train TD3 teachers using the standard 1M online steps [57, 69, 226], and store all encountered data tuples $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$ to make $\mathcal{D}^{(T)}$. Each training epoch consists of 4000 time steps, so there are 250 training epochs for teachers and students. In Section 7.5.2 we investigate training students for 10X more epochs. We apply standard noise levels for the teacher's exploration; the noise added to actions is $\mathcal{N}(0, 0.1)$ instead of $\mathcal{N}(0, 0.5)$ as done in some experiments in [3, 58] to increase data diversity. In the Appendix, we have results from SAC [69] teachers and students. The code we use is built on top of SpinningUp [2] from OpenAI.

Data Curriculum Experiments We test additive and scale curricula from Section 7.3.2. For additive curricula, we use $f \in \{50K, 100K, 200K, 500K, 1M\}$, which adjusts the "forward" samples allowed, and we also check if ignoring older samples in $\mathcal{D}^{(T)}$ helps (with $p = 800K$). We report scale curricula with $c \in \{0.50, 0.75, 1.00, 1.10, 1.25\}$, where $c < 1$

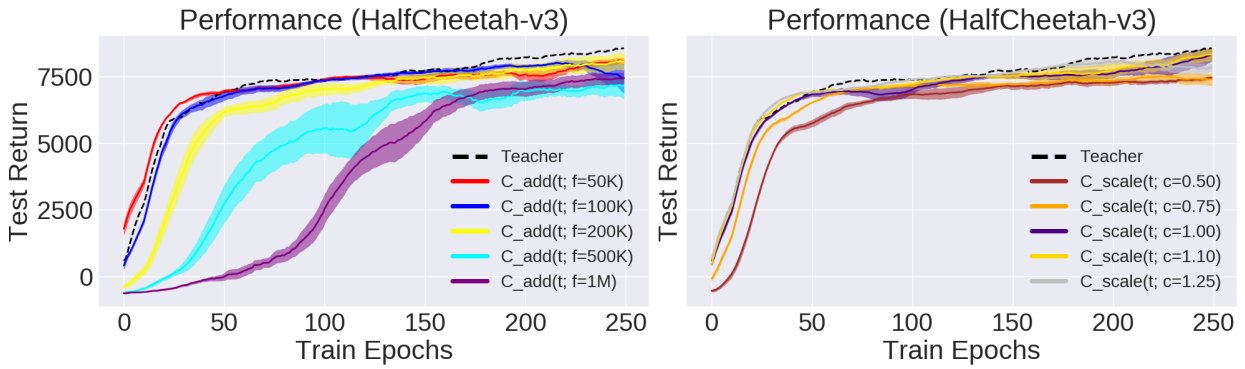

Figure 7.3: Offline student performance on HalfCheetah-v3 based on additive curricula (left) and scale curricula (right); these experiments correspond to numerical values in Table 7.1. In both subplots, the teacher's performance for reference (at *test* time, without noise) is shown in the dashed black line. Results from additive curricula suggest a clear pattern that access to more samples initially (*i.e.,* larger f) slows learning.

tests whether the student needs more gradient updates on data tuples in $\mathcal{D}^{(T)}$ relative to the teacher, and $c > 1$ tests whether it helps to have additional "forward" samples, which is similar to $f > 0$ in additive curricula, but where f increases throughout student training because the maximum eligible index $c \cdot t$ is a function of t. Due to computational limitations, for most experiments after Section 7.5.1, we test two particular curricula (from [58]): all the data ($C_{\text{add}}(t; f = 1M)$) or concurrent data ($C_{\text{scale}}(t; c = 1.00)$). We test with $C_{\text{add}}(t; f = 1M)$ because it serves a baseline of using all data, which can intuitively be viewed as using no curriculum. As Deep RL evaluations are notoriously noisy [79], all experiments are reproducible from code available on the project website.

Evaluation To evaluate student and teacher performance, we use two metrics, "M1" and "M2," defined as:

- **M1 (Final)**: the average reward of the student's final 100 test-time episodes.
- **M2 (Average)**: the average reward across *all* student test-time episodes.

In all experiments, students and teachers do 10 test episodes after every epoch. We use 5 random seeds for students, with respect to *one* teacher (per environment), to reduce the source of variability that would result from different teachers. We compute M1 and M2 statistics for each of the 5 runs, then average those 5 to get final numbers for M1 and M2. In the tables, when comparing a relevant set of students, we bold the best M1 and M2 results, *and additionally* bold other results with overlapping standard errors for a fairer comparison; see the Appendix for the exact formula.

Table 7.1: **Offline RL Results with DCUR**. Student performance as a function of 11 data curricula, with teacher performance (bottom row) as a reference. We report the "M1" and "M2" evaluation metrics (Section 7.4). We run 1 teacher per environment, then use 5 random seeds for training students from that same teacher data. Hence, all numbers reported below for student data curriculum experiments are averages over 5 independent offline RL runs; standard error values are in the Appendix. For each column, values are bolded for the best M1 and M2 results among *all* 11 curricula for students, *and* for other students with overlapping standard errors. The bolded values do not consider teacher performance, which is only present for a reference.

	Ant-v3		HalfCheetah-v3		Hopper-v3		Walker2d-v3	
Data Curriculum	M1	M2	M1	M2	M1	M2	M1	M2
$C_{\text{add}}(t; f = 50K)$	219.9	382.0	**8067.3**	**7147.0**	2986.2	1669.5	**2715.6**	**1712.1**
$C_{\text{add}}(t; f = 100K)$	-604.2	-409.6	7416.2	**7052.6**	2627.4	1980.1	**2746.7**	**1810.5**
$C_{\text{add}}(t; f = 200K)$	288.4	-620.4	**8028.4**	6521.5	2525.3	1887.9	**2691.8**	**1651.9**
$C_{\text{add}}(t; f = 500K)$	283.3	-801.1	7108.7	5041.2	2498.0	1546.2	1515.7	1183.5
$C_{\text{add}}(t; f = 1M)$	-1552.1	-1390.5	7447.3	3846.8	2323.0	1610.3	1741.4	1096.4
$C_{\text{add}}(t; p = 800K, f = 0)$	-889.7	1497.9	**7650.4**	6942.7	2132.2	1924.4	792.0	1521.8
$C_{\text{scale}}(t; c = 0.50)$	285.0	301.4	7467.8	6261.9	2332.3	1450.5	1866.0	799.9
$C_{\text{scale}}(t; c = 0.75)$	167.0	412.2	7392.6	6689.2	**3284.7**	1818.2	1864.9	1251.5
$C_{\text{scale}}(t; c = 1.00)$	825.6	1103.7	**8305.3**	6980.0	2984.3	2092.6	2178.5	1305.7
$C_{\text{scale}}(t; c = 1.10)$	**2952.5**	**2212.0**	**8306.0**	**7095.6**	**3185.2**	**2317.1**	2423.9	**1698.1**
$C_{\text{scale}}(t; c = 1.25)$	-1851.1	199.6	7843.8	**7175.4**	2755.5	1891.9	**2839.4**	**1747.4**
TD3 Teacher	4876.2	3975.8	8573.6	7285.5	3635.2	2791.9	3927.9	2579.8

7.5 Results

We present experimental results of DCUR under various settings, and report M1 and M2 metrics.

7.5.1 Effect of Data Curricula on Student Training Offline

With one fixed TD3 teacher data $\mathcal{D}^{(T)}$ per environment, we train students offline using 11 data curricula and list results in Table 7.1. The overall results suggest that a curriculum allowing for the available samples from $\mathcal{D}^{(T)}$ to grow over time, and to include a few samples "ahead" relative to the student's training time t produces stronger results. In particular, $C_{\text{scale}}(t; c = 1.10)$ has the best results, as it obtains the top scores or close to it (*i.e.,* with overlapping standard error) on 7 out of the 8 columns in Table 7.1. The next best curriculum, with 4 out of 8 top scores, is $C_{\text{add}}(t; f = 50K)$, and it similarly allows the available range of samples to go slightly past t, in this case by a fixed $t + 50K$. In contrast, allowing too much data with $C_{\text{add}}(t; f = 1M)$ or $C_{\text{add}}(t; f = 500K)$ results in poor performance, with neither of these curricula ranking among the best in any of the environments with respect to either metric. Ignoring older samples from the teacher's early training history with

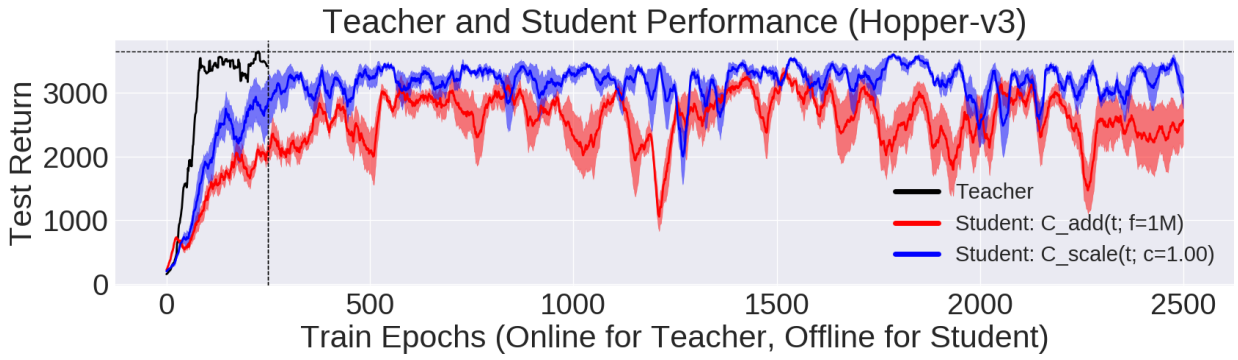

Figure 7.4: Hopper-v3 test-time episodic returns comparing the teacher performance (black curve) over its training period of 250 epochs (dashed vertical line), versus students trained offline over 2500 (10X more) epochs. For the two data curricula, we train 5 independent students from the fixed teacher data $\mathcal{D}^{(T)}$ with different seeds. Results suggest that the $C_{\text{scale}}(t; c = 1.00)$ curriculum (blue curve) can lead to better average performance over 2500 epochs, and that it matches the teacher's best performance (dashed horizontal line).

Table 7.2: **Offline RL Results with DCUR, 10X Training**. Student performance as a function of the data curriculum, where the students train for 10X longer (2500 epochs) as compared to student results in Table 7.1. Besides this one change, the setup and table formatting is identical to Table 7.1. See Section 7.5.2 for details. The teacher metrics are repeated for reference, and are not considered when bolding numbers in the table.

	Ant-v3		HalfCheetah-v3		Hopper-v3		Walker2d-v3	
Curriculum	M1	M2	M1	M2	M1	M2	M1	M2
$C_{\text{add}}(t; f = 1M)$	**-2770.0**	-2088.3	**6041.6**	6847.9	**2593.1**	2482.0	**3713.1**	2763.5
$C_{\text{scale}}(t; c = 1.00)$	**-2780.6**	**-1511.2**	**6496.2**	**7555.5**	**3019.4**	**3052.3**	**3208.0**	**3121.4**
TD3 Teacher	4876.2	3975.8	8573.6	7285.5	3635.2	2791.9	3927.9	2579.8

$C_{\text{add}}(t; p = 800K, f = 0)$ also exhibits weak performance. See Figure 7.3 for a representative set of learning curves for HalfCheetah-v3, showing some curricula that result in the student essentially matching teacher performance, despite known challenges associated with pure offline learning, even with relying on concurrent-style training [58].

7.5.2 Data Curriculum for Training 10X Longer

Here, as in Section 7.5.1, we run the $C_{\text{add}}(t; f = 1M)$ and $C_{\text{scale}}(t; c = 1.00)$ curricula again, but train 10X longer to understand how this affects learning. The choice of curriculum will *only* affect the first 1/10 of training (*i.e.,* the first 250 epochs), since after that, both curricula

Table 7.3: **DCUR Results with Online Data.** Performance of student learning based on one of two data curriculum choices and the addition of a small fraction of online data. At the end of 250 training epochs, the student has collected self-generated, online data that constitutes 2.5%, 5.0%, or 10.0% of the original teacher data size $|\mathcal{D}^{(T)}| = 1M$. We use the same teacher data as in Table 7.1 and report the M1 and M2 metrics. We bold values by comparing only the two curricula with the same X% online experiments per column and bolding the maximum only (if non-overlapping standard error) or both (if otherwise).

	Ant-v3		HalfCheetah-v3		Hopper-v3		Walker2d-v3	
Curriculum; % Online	M1	M2	M1	M2	M1	M2	M1	M2
$C_{\text{add}}(t; f = 1M)$; 2.5%	3093.1	894.4	**8581.7**	5275.3	**3354.2**	1977.9	**3298.1**	1882.6
$C_{\text{scale}}(t; c = 1.00)$; 2.5%	**4004.7**	**2857.0**	**8417.9**	**7212.2**	2712.9	**2238.1**	**3144.9**	**2050.3**
$C_{\text{add}}(t; f = 1M)$; 5.0%	**3693.9**	1556.6	**8658.8**	5634.6	**3232.1**	2230.5	**3243.8**	**2097.3**
$C_{\text{scale}}(t; c = 1.00)$; 5.0%	**3251.1**	**3068.2**	**8601.4**	**7274.0**	**3356.2**	**2459.0**	**3155.8**	**2097.0**
$C_{\text{add}}(t; f = 1M)$; 10.0%	4229.0	2007.6	**8864.2**	6024.5	**2741.2**	1979.4	**3607.7**	**2448.2**
$C_{\text{scale}}(t; c = 1.00)$; 10.0%	**4510.9**	**3349.0**	8608.5	**7290.5**	**3182.9**	**2580.6**	3146.9	2204.2
TD3 Teacher	4876.2	3975.8	8573.6	7285.5	3635.2	2791.9	3927.9	2579.8

reduce to the student sampling data tuples from the entire teacher buffer $\mathcal{D}^{(T)}$. To make results comparable with those in Section 7.5.1, we use the same teacher buffers. Table 7.2 has results in a similar manner as Table 7.1. We find that, somewhat surprisingly, the curricula affects the long-term performance of the student in the remaining 9/10 of training. Across all 8 columns (4 environments and the M1/M2 metrics), using $C_{\text{scale}}(t; c = 1.00)$ outperforms $C_{\text{add}}(t; f = 1M)$ or is statistically similar to it based on standard errors, suggesting that the initial curriculum assists TD3 in finding a stable set of policy and value functions so that it can continue training without significant deterioration. Figure 7.4 shows a representative set of learning curves for Hopper-v3, which shows the student with $C_{\text{scale}}(t; c = 1.00)$ matching the teacher's performance given sufficient training.

7.5.3 Apprenticeship Learning: Results with Small Amounts of Online Data

We next study when students can use a small amount of online data to address challenges with pure offline learning. We train students for 250 epochs using 2.0%, 5.0% and 10.0% online data collection, *i.e.,* at the *end* of 1M training steps, students get 25K, 50K, and 100K self-generated data tuples, respectively, for $\mathcal{D}^{(S)}$, which they can sample from in addition to the (filtered) data tuples from $\mathcal{D}^{(T)}$. Table 7.3 contains the M1 and M2 results across all 4 environments. The results indicate that even with as little as 2.5% online data, students are able to significantly improve versus offline learning, with the improvement most notable in the complex Ant-v3 environment as shown in Figure 7.5. Furthermore, $C_{\text{scale}}(t; c = 1.00)$

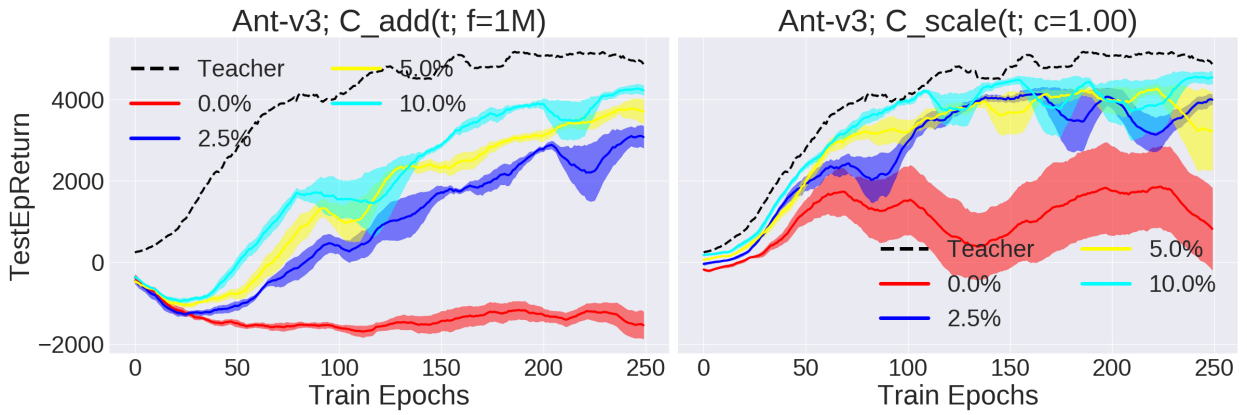

Figure 7.5: Ant-v3 test-time performance for students with various amounts of online data, ranging from 0% (*i.e.,* offline) to 10% online. We show the teacher curve (dashed black line) for reference. We plot results from two curricula, $C_{\text{add}}(t; f = 1M)$ to the left, and $C_{\text{scale}}(t; c = 1.00)$ to the right.

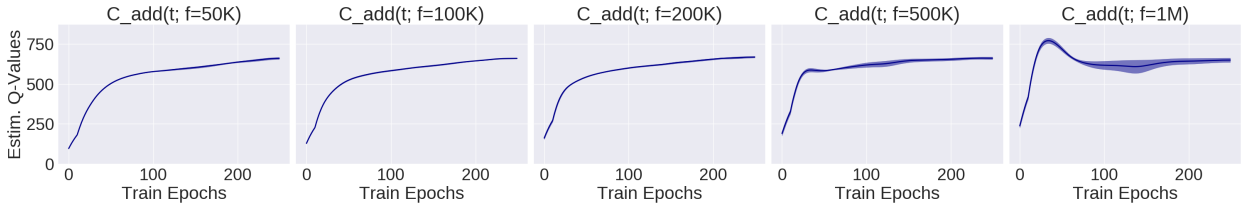

Figure 7.6: Students' estimated Q-values in HalfCheetah-v3, as a function of training epoch. These are from students in Table 7.1. With more data available at the start of training (*i.e.,* increasing f, shown left to right) this creates initial over-estimation of Q-values. All 5 subplots are derived from students reported in Table 7.1, and all curves average over the 5 random seeds and show standard errors.

continues to provide some benefit to learning speed compared to $C_{\text{add}}(t; f = 1M)$ with online data, particularly with respect to M2.

7.5.4 Investigation of Q-Values

To understand why the data curriculum matters, we investigate the student's estimated Q-values. With too much data available from $\mathcal{D}^{(T)}$ during early stages of training, this causes overestimation of Q-values, whereas a curriculum that restricts data tuples results in more stable, monotonically increasing Q-values. Figure 7.6 shows different additive curricula on HalfCheetah-v3 and plots the estimated Q-values. A similar trend holds for the other environments.

7.6 Conclusion and Future Work

This chapter introduces the DCUR framework, which studies how to filter a given dataset for existing reinforcement learning algorithms. Results across a variety of curricula and training settings suggest that the choice significantly impacts the learning speed of students running RL. The greatest benefits come from curricula that strictly limit the availability of data tuples at the start, but gradually lets the available data grow throughout training. We caution that the results presented are contingent on the way we generated the datasets, *i.e.,* the logged training history of an RL algorithm. In future work, a priority is to expand Tables 7.2 and 7.3 with results from additional curricula, and to better quantify how much self-generated student data is necessary for the student to surpass teacher performance.

In addition, we will test other datasets and environments, such as those from D4RL [55] or RL Unplugged [67]. We will explore learning from multiple teachers [115]. While we kept relevant experience replay hyperparameters such as the *replay ratio* constant, we will make use of findings from recent research [48] to further understand the interplay between experience replay and data curriculum strategies. Finally, we plan to devise and test more sophisticated data curricula.

Part V

Conclusion and Future Work

Chapter 8

Conclusion

This dissertation has presented a summary of work at the intersection of robotics, computer vision, and machine learning, with much of it focusing on goal-driven research[1] for deformable object manipulation. In Chapter 2, we presented a system that could perform a simplified version of robot bed-making on a quarter-scale bed. Next, in Chapter 3, we attempted to address two weaknesses of the prior system: the requirement for real world data collection and the inflexibility associated with a fixed place point (for the "pick-and-place" action). This motivated the development of a custom fabric simulator for smoothing, and we performed simulation-to-real transfer on a physical da Vinci Surgical Robot. This work then motivated Chapter 4, where we tried to generalize this approach to goal-conditioned fabric manipulation, so that the policy is not fixed to a specific task (*e.g.,* smoothing). These systems deal with fabric manipulation in different ways, each with fundamental tradeoffs in data collection, policy flexibility, and deployment speed.

The various deformable object simulation tasks from these chapters then led us to consider two main questions: (1) is it possible to develop a standardized benchmark for deformable object simulation so that other researchers can use it, which will help better measure progress in the field, and (2) can we move beyond fabric smoothing and folding, and consider a more complex task which might require reasoning about 3D space, such as bag manipulation? Chapter 5 presents research inspired by both questions. We open-source a benchmark for deformable simulation, and include several tasks that involve manipulating highly deformable 3D bags containing items.

Finally, Chapters 6 and 7 take a step back and consider robotics more broadly: if we are learning from demonstrations or from a large offline dataset, which we have done in all the presented fabric manipulation work, what are ways that we might be able to leverage this supervision, depending on the structure of the data? We consider when a robot (*i.e.,* "student") can obtain samples from a logged training history of a reinforcement learning agent (*i.e.,* "teacher"), and we show performance gains by limiting the range of allowed samples that a student agent can utilize. In ongoing work, we are applying the ideas explored

[1] `http://joschu.net/blog/opinionated-guide-ml-research.html`

in these works for physical fabric and bag manipulation.

We hope that this dissertation inspires new research directions. We next discuss what we believe are highly promising avenues for future work.

8.1 Future Research Opportunities

New Robotics Tasks One line of thinking in robotics research is to get robots to perform a novel task. The challenges are ensuring that the task is interesting and set at the right research scope and difficulty. For example, Chapter 2 was directly inspired by a novel task: robot bed-making, which we believed might be a good fit for automation since it is common, rarely enjoyed, not time critical, and tolerant to slight errors. Similarly, Chapter 5 was inspired by manipulating bags and inserting items in them. The process of devising methods to tackle these novel tasks can additionally generate new ideas of independent interest. For example, while bag manipulation was the original idea for the work in Chapter 5, during the process of this research, we developed a novel neural network architecture for goal-conditioned object rearrangement. A closely-related research direction is to take an existing task-driven robotics research result, and significantly improve performance, or show how one can alleviate strong assumptions of a particular setup or experiment.

In future work, I hope to see the research community expand the frontiers of tasks which robots can perform autonomously. Some ideas for tasks that pique my research curiosity include wrapping, cooking, cutting [178, 77], manipulating dough [141], manipulating granular media [180], more advanced (physical) bag management, as well as surgical tasks such as suturing [189]. In this dissertation, while we reported experiments using the da Vinci Research Kit as a testbed for fabric smoothing as in Chapters 3 and 4, I was also fortunate to work on tasks specific to surgical robotics, such as surgical peg transfer [89, 90, 91, 157] and surgical debridement [186]. Such work builds upon a rich literature in robot surgery [231, 34], and given the nature of surgery which operates on organic tissue matter, these tasks also fall under the umbrella of deformable object manipulation.

The preceding citations show that there have been preliminary results in some of these directions, but there is likely much more progress that can be done and numerous variations of the existing tasks. The world presents a seemingly infinite supply of new tasks to try – the possibilities are endless!

Better Simulators In this dissertation, a recurring theme is the use of simulators, particularly Blender [28] and PyBullet [30]. In Chapters 3 and 4, we used a custom-designed simulator [184] for fabric smoothing and folding, and used domain randomization [207] for simulation-to-real transfer. In Chapter 5, we developed a simulation benchmark built on PyBullet for deformable manipulation. The use of simulators, however, should be justified. The 2020 *Robotics: Science and Systems (RSS)* conference featured a lively debate over whether investing in simulation-to-real learning is a waste of time.[2] As might be evident from this

[2] `https://sim2real.github.io/`

dissertation, I have personally been in favor of simulation-to-real learning. Using simulators can let us avoid tedious manual data collection (as we had to do in Chapter 2), facilitates reproducibility and benchmarks in robotics, makes it easier to compare multiple algorithms before deploying them in real, and may help aspiring researchers break into the field if they lack of access to physical robots. Given the ongoing COVID-19 pandemic, working with simulators also reduces the need to enter physical lab settings.

The major downside of simulators is that they never perfectly model real-world physics, a phenomenon colloquially referred to as the "reality gap," and researchers can expend massive computational resources training useless policies in simulation. While this gap exists for any simulator, different simulators have different fidelities, so one area of research is to further refine and improve simulator fidelity, under the theory that learning policies from more accurate simulators better facilitates simulation-to-real transfer. Empirically, I have noticed that NVIDIA FleX, the physics backend in SoftGym [128], can simulate layers of fabrics significantly better than PyBullet or the simulator that I helped to design, both of which lead to unrealistic overlaps and self-collisions. More recently, Incremental Potential Contact (IPC) [122] has emerged as a promising, high-fidelity simulator. While these advances are exciting, there is another subtle issue at play: high-fidelity simulators such as IPC and ARCSim [150] are often developed by computer graphics researchers, who have a different set of needs compared to data-driven robotics researchers who may prioritize simulator speed.[3] In future work, it will be exciting to see if the communities of (deformable object) simulation developers and roboticists combine their efforts. This might additionally allow for accurate simulation of *grasping* deformables. In my work, and others that use fabric simulators [227, 229, 128, 68], it is common to assume fake grippers which can "pin" or "fix" vertices within some radius to the gripper. In future work, it would be nice to alleviate this assumption by simulating a parallel-jaw gripper handling a deformable, with friction modeled between layers of fabric and the gripper.

Moving Beyond Quasistatic Pick-and-Place All my robotic manipulation work presented in this dissertation relies on a pick-and-place action space in a quasistatic setting, meaning that the objects of interest are allowed to settle in between actions. This action space is sufficient for a variety of tasks and its low dimensionality makes it easier to do data-driven learning, but it might seem strange when comparing robotic manipulation with human manipulation. For example, as discussed in Chapter 5, a quasistatic pick-and-place action space means the robot cannot easily recover from failures that happen during action execution. In the bag manipulation tasks, the simulated UR5 robot frequently ran into failures when it was pulling the bag upwards but had some items fall out of the bag. A human would naturally stop the motion entirely or perhaps attempt to dynamically adjust the bag to contain its item(s) in case any might be close to falling out. Similarly, as highlighted in a recent result on real-world dynamic fabric manipulation [68], humans rarely do a task like

[3]The RSS 2021 workshop on Deformable Object Simulation in Robotics included a overview of this tradeoff: `https://sites.google.com/nvidia.com/do-sim/home`.

bed-making by engaging in slow, pick-and-place actions on the blanket; we are more likely to perform a dynamic "tossing" or "flinging" action.

In future work, I am interested in developing higher-rate control policies so that robots can react and potentially address failures in real-time. One option is to incorporate continuous visual servoing. While it is possible to train a policy that can output an action (*e.g.,* a delta in coordinate space or motor torques) learned from image pixels at a high frequency rate, this could require significant amounts of training data beyond the reach of all but the most resourceful research labs [120, 96]. Exploring how to train such continuous visual servoing for closed-loop manipulation in a data-efficient manner could be a promising avenue for future research. It would also be interesting to introduce or extend challenging manipulation tasks that require moving away from pick-and-place entirely, as exemplified with research on dynamic manipulation such as tossing [239], casting cables [241], or flinging fabric [68].

Representations for Deformable Objects What is the right representation to use for deformable object manipulation? As I mention in Chapter 1, I view this as deciding on how the data should be provided to or utilized in a machine learning system. For a deformable object with a configuration that is difficult to specify with a finite set of 6-DoF poses, perhaps the most obvious representation one can use is to simply pass raw images of the deformable directly to a deep neural network policy to produce an action. In Chapters 2, 3, and 4, we represented the input with standard RGBD, RGB, and D images, and were able to attain real-world fabric manipulation. However, there might be ways to design better representations that can improve sample efficiency.

For example, Yan et al. [229] pursue a model-based approach for fabric manipulation as we do in Chapter 4. While we plan in image space, though, Yan et al. use contrastive learning to plan in a *latent space*, since learning a video prediction model on raw images can be challenging. Enforcing this representation means that the distance between latent vectors can efficiently encode the "closeness" of two different fabric configurations. Another option for fabric manipulation is to use a flow-based representation. In FabricFlowNet [5], the proposed system uses optical flow to perform image-based, goal-conditioned fabric manipulation. Given the current and goal images of a fabric, the system predicts the optical flow between the two images, and then passes that into the policy to determine picking actions (with the placing actions determined directly from optical flow). An alternative to a flow-based representation is to use a particle-based representation to explicitly reason about and predict (visible) parts of a fabric [127]. Finally, one can segment an image of a fabric before passing it to a policy to identify key regions for grasping [166].

Other representations can be *action-centric*, such as with the recent innovation of Transporter Networks [240] and their Goal-Conditioned variant (covered in Chapter 5), which use the idea of "transporting" feature image crops to find good placing points on an underlying image, and which are remarkably sample efficient for top-down pick-and-place rearrangement. These are "action-centric" representations because the convolution operator performs a brute-force search over all placing spots and encodes all possible placing actions.

Overall, I am excited to see if there are other representations that can be useful for robotic manipulation, and how far we can push existing representations for other tasks.

Multi-Modal Sensing for Robotics In this dissertation, I have primarily focused on learning policies based on color and/or depth images (for robotic manipulation and for Chapter 6) or state-based input from joint angle positions and velocities (for Chapter 7). While this information can be sufficient to tackle many robotics tasks, an exciting area of future work is to extend the capabilities of robots by allowing them to utilize multi-modal sensing. In robot manipulation, the sense of touch can be a valuable tool. Recent advances in tactile sensors, such as with GelSight [234] and GelSlim [204], suggest that it might be possible to learn manipulation from touch information and reinforcement learning [36]. Similarly, audio can be another source of information for robotic manipulation [62, 142]. These different sensory modes has also motivated a line of research on how to best fuse these sources of supervision [117]. In future work, I hope to see the robotics research community extend the capabilities of robots dealing with multiple types of data sources.

8.2 The Long-Term Vision

What will be the future of robotics? The last decade has seen exciting progress in learning-based approaches for robotics, but we have arguably not yet seen widespread transfer of robots from research laboratories to the messy and unstructured real world. I am hopeful that this dissertation can play a small part in opening the doors for robots to be deployed throughout homes, buildings, and the environment, for the betterment of society. One of my most exciting thoughts is to envision a fleet of robots continually learning to manipulate the tangled, complex, and diverse set of objects that our glorious world provides.

I am unsure as to whether this will be possible in my lifetime, but I will do my best to turn this vision into a reality.

... until next time!

Daniel Seita. Berkeley, CA.

Bibliography

[1] Pieter Abbeel and Andrew Ng. "Apprenticeship Learning via Inverse Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2004.

[2] Joshua Achiam. *Spinning Up in Deep Reinforcement Learning*. `https://github.com/openai/spinningup`. 2018.

[3] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. "An Optimistic Perspective on Offline Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2020.

[4] Marcin Andrychowicz et al. "Hindsight Experience Replay". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[5] Anonymous. "FabricFlowNet: Bimanual Cloth Manipulation with a Flow-based Policy". In: *Submitted to 5th Annual Conference on Robot Learning*. under review. 2021. URL: `https://openreview.net/forum?id=TsqkJJMgHkk`.

[6] Brenna D Argall et al. "A Survey of Robot Learning From Demonstration". In: *Robotics and Autonomous Systems* 57 (2009).

[7] Mohammad Babaeizadeh et al. "Stochastic Variational Video Prediction". In: *International Conference on Learning Representations (ICLR)*. 2018.

[8] Yunfei Bai, Wenhao Yu, and C. K. Liu. "Dexterous Manipulation of Cloth". In: *European Association for Computer Graphics*. 2016.

[9] Benjamin Balaguer and Stefano Carpin. "Combining Imitation and Reinforcement Learning to Fold Deformable Planar Objects". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.

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

[11] David Baraff and Andrew Witkin. "Large Steps in Cloth Simulation". In: *ACM SIGGRAPH*. 1998.

[12] K.J. Bathe. *Finite Element Procedures*. Prentice Hall, 2006. URL: `https://books.google.com/books?id=rWvefGICfO8C`.

[13] Dhruv Batra et al. "Rearrangement: A Challenge for Embodied AI". In: *arXiv preprint arXiv:2011.01975* (2020).

[14] Jenay M Beer et al. "The Domesticated Robot: Design Guidelines for Assisting Older Adults to Age in Place". In: *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 2012.

[15] M. G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *Journal of Artificial Intelligence Research* (2013).

[16] Yoshua Bengio et al. "Curriculum Learning". In: *International Conference on Machine Learning (ICML)*. 2009.

[17] Felix Berkenkamp, Angela P Schoellig, and Andreas Krause. "Safe Controller Optimization for Quadrotors with Gaussian Processes". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[18] Christopher M. Bishop. "Mixture Density Networks". In: *Aston University* (1994).

[19] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[20] Jacqueline Bloomfield, Anne Pegram, and Anne Jones. "Recommended Procedure for Bedmaking in Hospital". In: *Nursing Standard* 22 (2008).

[21] Julia Borras, Guillem Alenya, and Carme Torras. "A Grasping-centered Analysis for Cloth Manipulation". In: *arXiv preprint arXiv:1906.08202* (2019).

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

[23] Yuri Burda et al. "Exploration by Random Network Distillation". In: *International Conference on Learning Representations (ICLR)*. 2019.

[24] Seth Chaiklin. "The Zone of Proximal Development in Vygotsky's Analysis of Learning and Instruction". In: *Vygotsky's Educational Theory in Cultural Context* (2003).

[25] Dian Chen et al. "Learning by Cheating". In: *Conference on Robot Learning (CoRL)*. 2019.

[26] A. Chiuso and G. Pillonetto. "System Identification: A Machine Learning Perspective". In: *Annual Review of Control, Robotics, and Autonomous Systems* (2019).

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

[28] Blender Online Community. *Blender – a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: `http://www.blender.org`.

[29] E. Corona et al. "Active Garment Recognition and Target Grasping Point Detection Using Deep Learning". In: *Pattern Recognition*. 2018.

[30] Erwin Coumans and Yunfei Bai. *PyBullet, a Python Module for Physics Simulation for Games, Robotics and Machine Learning*. `http://pybullet.org`. 2021.

[31] Marco Cusumano-Towner et al. "Bringing Clothing Into Desired Configurations with Limited Perception". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2011.

[32] Wojciech Marian Czarnecki et al. "Mix & Match - Agent Curricula for Reinforcement Learning". In: *International Conference on Machine Learning (ICML)*. 2018.

[33] Sudeep Dasari et al. "RoboNet: Large-Scale Multi-Robot Learning". In: *Conference on Robot Learning (CoRL)*. 2019.

[34] Claudia DEttorre et al. "Accelerating Surgical Robotics Research: Reviewing 10 Years of Research with the dVRK". In: *arXiv preprint arXiv:2104.09869* (2021).

[35] Prafulla Dhariwal et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[36] Siyuan Dong et al. "Tactile-RL for Insertion: Generalization to Objects of Unknown Geometry". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[37] Andreas Doumanoglou et al. "Autonomous Active Recognition and Unfolding of Clothes Using Random Decision Forests and Probabilistic Planning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

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

[39] Jeffrey Elman. "Learning and Development in Neural Networks: The Importance of Starting Small". In: *Cognition* (July 1993).

[40] Zackory Erickson et al. "Assistive Gym: A Physics Simulation Framework for Assistive Robotics". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[41] Zackory Erickson et al. "Deep Haptic Model Predictive Control for Robot-Assisted Dressing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[42] Zackory Erickson et al. "Tracking Human Pose During Robot-Assisted Dressing using Single-Axis Capacitive Proximity Sensing". In: *IEEE Robotics and Automation Letters (RA-L)*. 2018.

[43] Damien Ernst, Pierre Geurts, and Louis Wehenkel. "Tree-based Batch Mode Reinforcement Learning". In: *Journal of Machine Learning Research* (2005).

[44] M. Everingham et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/. 2012.

[45] Neta Ezer, Arthur Fisk, and Wendy Rogers. "More than a Servant: Self-Reported Willingness of Younger and Older Adults to having a Robot perform Interactive and Critical Tasks in the Home". In: *Proc Hum Factors Ergon Soc Annu Meet* (2009).

[46] Yang Fan et al. "Learning to Teach". In: *International Conference on Learning Representations (ICLR)*. 2018.

[47] Cara Bailey Fausset et al. "Challenges to Aging in Place: Understanding Home Maintenance Difficulties". In: *Journal of Housing for the Elderly* 25.2 (2011), pp. 125–141.

[48] William Fedus et al. "Revisiting Fundamentals of Experience Replay". In: *International Conference on Machine Learning (ICML)*. 2020.

[49] Chelsea Finn and Sergey Levine. "Deep Visual Foresight for Planning Robot Motion". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[50] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981).

[51] Pete Florence, Lucas Manuelli, and Russ Tedrake. "Dense Object Nets: Learning Dense Visual Object Descriptors By and For Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. 2018.

[52] Carlos Florensa et al. "Automatic Goal Generation for Reinforcement Learning Agents". In: *International Conference on Machine Learning (ICML)*. 2018.

[53] Carlos Florensa et al. "Reverse Curriculum Generation for Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. 2017.

[54] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. "Maneuver-based Motion Planning for Nonlinear Systems with Symmetries". In: *IEEE Transactions on Robotics (T-RO)* (2005).

[55] Justin Fu et al. "D4RL: Datasets for Deep Data-Driven Reinforcement Learning". In: *arXiv preprint arXiv:2004.07219* (2020).

[56] Justin Fu et al. "Diagnosing Bottlenecks in Deep Q-learning Algorithms". In: *International Conference on Machine Learning (ICML)*. 2019.

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

[58] Scott Fujimoto, David Meger, and Doina Precup. "Off-Policy Deep Reinforcement Learning without Exploration". In: *International Conference on Machine Learning (ICML)*. 2019.

[59] Scott Fujimoto et al. "Benchmarking Batch Deep Reinforcement Learning Algorithms". In: *arXiv preprint arXiv:1910.01708* (2019).

[60] Tommaso Furlanello et al. "Born Again Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2018.

[61] 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.

[62] Dhiraj Gandhi, Abhinav Gupta, and Lerrel Pinto. "Swoosh! Rattle! Thump! – Actions that Sound". In: *Robotics: Science and Systems (RSS)*. 2020.

[63] Yixing Gao, Hyung Jin Chang, and Yiannis Demiris. "Iterative Path Optimisation for Personalised Dressing Assistance using Vision and Force Information". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

[64] Kenneth Goldberg. "Stochastic Plans for Robot Manipulation". PhD thesis. Carnegie Mellon University, 1990.

[65] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[66] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Neural Information Processing Systems (NeurIPS)*. 2014.

[67] Caglar Gulcehre et al. "RL Unplugged: A Suite of Benchmarks for Offline Reinforcement Learning". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[68] Huy Ha and Shuran Song. "FlingBot: The Unreasonable Effectiveness of Dynamic Manipulation for Cloth Unfolding". In: *arXiv preprint arXiv:2105.03655* (2021).

[69] 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.

[70] Martin Hägele et al. "Industrial Robotics". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Springer International Publishing, 2016, pp. 1385–1422.

[71] Chris Harris and Mike Stephens. "A Combined Corner and Edge Detector". In: *In Proceedings of the Fourth Alvey Vision Conference*. 1988.

[72] Kunimatsu Hashimoto et al. "A Field Study of the Human Support Robot in the Home Environment". In: *IEEE Workshop on Advanced Robotics and its Social Impacts*. 2013.

[73] Hado van Hasselt, Arthur Quez, and David Silver. "Deep Reinforcement Learning With Double Q-Learning". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2016.

[74] Naohiro Hayashi, Takashi Suehiro, and Shunsuke Kudoh. "Planning Method for a Wrapping-With-Fabric Task Using Regrasping". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[75] Naohiro Hayashi et al. "Dual Arm Robot Fabric Wrapping Operation Using Target Lines". In: *IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2014.

[76] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[77] Eric Heiden et al. "DiSeCT: A Differentiable Simulation Engine for Autonomous Robotic Cutting". In: *Robotics: Science and Systems (RSS)*. 2021.

[78] Randall B Hellman et al. "Functional Contour-following via Haptic Perception and Reinforcement Learning". In: *IEEE Transactions on Haptics*. 2018.

[79] Peter Henderson et al. "Deep Reinforcement Learning that Matters". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2018.

[80] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2018.

[81] Todd Hester et al. "Deep Q-learning from Demonstrations". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2018.

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

[83] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the Knowledge in a Neural Network". In: *NeurIPS Deep Learning Workshop*. 2014.

[84] J. Hopcroft, J. Kearney, and D. Krafft. "A Case Study of Flexible Object Manipulation". In: *International Journal of Robotics Research (IJRR)*. 1991.

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

[86] Ryan Hoque et al. "VisuoSpatial Foresight for Physical Sequential Fabric Manipulation". In: *arXiv preprint arXiv:2102.09754*. 2021.

[87] Dan Horgan et al. "Distributed Prioritized Experience Replay". In: *International Conference on Learning Representations (ICLR)*. 2018.

[88] Ayanna Howard and George Bekey. "Intelligent Learning for Deformable Object Manipulation". In: *Autonomous Robotics*. 2000.

[89] Minho Hwang et al. "Applying Depth-Sensing to Automated Surgical Manipulation with a da Vinci Robot". In: *International Symposium on Medical Robotics (ISMR)*. 2020.

[90] Minho Hwang et al. "Efficiently Calibrating Cable-Driven Surgical Robots With RGBD Sensing, Temporal Windowing, and Linear and Recurrent Neural Network Compensation". In: *IEEE Robotics and Automation Letters (RA-L)*. 2020.

[91] Minho Hwang et al. "Superhuman Surgical Peg Transfer Using Depth-Sensing and Deep Recurrent Neural Networks". In: *arXiv preprint arXiv:2012.12844* (2020).

[92] Allan Jabri et al. "Unsupervised Curricula for Visual Meta-Reinforcement Learning". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[93] Rishabh Jangir, Guillem Alenya, and Carme Torras. "Dynamic Cloth Manipulation with Deep Reinforcement Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[94] Biao Jia et al. "Cloth Manipulation Using Random-Forest-Based Imitation Learning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[95] Biao Jia et al. "Manipulating Highly Deformable Materials Using a Visual Feedback Dictionary". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

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

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

[98] H Kazerooni and Chris Foley. "A Robot Mechanism for Grapsing Sacks". In: *IEEE Transactions on Automation Science and Engineering*. 2005.

[99] Rahul Kidambi et al. "MOReL: Model-Based Offline Reinforcement Learning". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[100] Chih-Hung King et al. "Towards an Assistive Robot that Autonomously Performs Bed Baths for Patient Hygiene". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010.

[101] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2015.

[102] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2015.

[103] Alice Kirchheim, Matthias Burwinkel, and Wolfgang Echelmeyer. "Automatic Unloading of Heavy Sacks From Containers". In: *IEEE International Conference on Automation and Logistics (ICAL)*. 2008.

[104] Yasuyo Kita et al. "A Method For Handling a Specific Part of Clothing by Dual Arms". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2009.

[105] Yasuyo Kita et al. "Clothes State Recognition Using 3D Observed Data". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2009.

[106] Ellen Klingbeil et al. "Grasping With Application to an Autonomous Checkout Robot". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2011.

[107] Jus Kocijan et al. "Gaussian Process Model Based Predictive Control". In: *American Control Conference (ACC)*. 2004.

[108] Jan J Koenderink and Andrea J Van Doorn. "Surface shape and curvature scales". In: *Image and vision computing* 10.8 (1992).

[109] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems (NeurIPS)*. 2012.

[110] 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).

[111] Aviral Kumar, Abhishek Gupta, and Sergey Levine. "DisCor: Corrective Feedback in Reinforcement Learning via Distribution Correction". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[112] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[113] Aviral Kumar et al. "Implicit Under-Parameterization Inhibits Data-Efficient Deep Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2021.

[114] Aviral Kumar et al. "Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction". In: *Neural Information Processing Systems (NeurIPS)*. 2019.

[115] 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.

[116] Sascha Lange, Thomas Gabel, and Martin Riedmiller. "Batch Reinforcement Learning". In: *RL. Adaptation, Learning, and Optimization* (2012).

[117] Michelle A. Lee et al. "Making Sense of Vision and Touch: Self-Supervised Learning of Multimodal Representations for Contact-Rich Tasks". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[118] Robert Lee et al. "Learning Arbitrary-Goal Fabric Folding with One Hour of Real Robot Experience". In: *Conference on Robot Learning (CoRL)*. 2020.

[119] Sergey Levine et al. "End-to-end Training of Deep Visuomotor Policies". In: *Journal of Machine Learning Research (JMLR)*. 2016.

[120] 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)*. 2017.

[121] Sergey Levine et al. "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[122] Minchen Li et al. "Incremental Potential Contact: Intersection- and Inversion-free Large Deformation Dynamics". In: *ACM Trans. Graph. (SIGGRAPH)* 39.4 (2020).

[123] Yinxiao Li et al. "Folding Deformable Objects using Predictive Simulation and Trajectory Optimization". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015.

[124] Yinxiao Li et al. "Multi-Sensor Surface Analysis for Robotic Ironing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[125] Timothy P. Lillicrap et al. "Continuous Control with Deep Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2016.

[126] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching". In: *Machine Learning* (1992).

[127] Xingyu Lin, Yufei Wang, and David Held. "Learning Visible Connectivity Dynamics for Cloth Smoothing". In: *arXiv preprint arXiv:2105.10389* (2021).

[128] Xingyu Lin et al. "SoftGym: Benchmarking Deep Reinforcement Learning for Deformable Object Manipulation". In: *Conference on Robot Learning (CoRL)*. 2020.

[129] 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 (IROS)*. 2020.

[130] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[131] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[132] Marlos C. Machado et al. "Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents". In: *arXiv preprint arXiv:1709.06009* (2017).

[133] 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)*. 2017.

[134] Jeffrey Mahler et al. "Learning Accurate Kinematic Control of Cable-Driven Surgical Robots Using Data Cleaning and Gaussian Process Regression." In: *IEEE Conference on Automation Science and Engineering (CASE)*. 2014.

[135] Jeffrey Mahler et al. "Learning Ambidextrous Robot Grasping Policies". In: *Science Robotics* 4.26 (2019).

[136] Jeremy Maitin-Shepard et al. "Cloth Grasp Point Detection Based on Multiple-View Geometric Cues with Application to Robotic Towel Folding". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010.

[137] H. B. Mann and D. R. Whitney. "On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other". In: *The Annals of Mathematical Statistics* (1947).

[138] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[139] Jan Matas, Stephen James, and Andrew J. Davison. "Sim-to-Real Reinforcement Learning for Deformable Object Manipulation". In: *Conference on Robot Learning (CoRL)* (2018).

[140] Tambet Matiisen et al. "Teacher-Student Curriculum Learning". In: *arXiv preprint arXiv:1707.00183* (2017).

[141] Carolyn Matl, Josephine Koe, and Ruzena Bajcsy. "StRETcH: a Soft to Resistive Elastic Tactile Hand". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[142] Carolyn Matl et al. "STReSSD: Sim-To-Real from Sound for Stochastic Dynamics". In: *Conference on Robot Learning (CoRL)*. 2020.

[143] Stephen Miller et al. "A Geometric Approach to Robotic Laundry Folding". In: *International Journal of Robotics Research (IJRR)*. 2012.

[144] Volodymyr Mnih et al. "Human-Level Control Through Deep Reinforcement Learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533.

[145] T. Morita et al. "Knot Planning from Observation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2003.

[146] Toshiharu Mukai et al. "Development of a Nursing-Care Assistant Robot RIBA That Can Lift a Human in Its Arms". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2010.

[147] Ashvin Nair et al. "Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

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

[149] Suraj Nair et al. "Time Reversal as Self-Supervision". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[150] Rahul Narain, Armin Samii, and James F. O'Brien. "Adaptive Anisotropic Remeshing for Cloth Simulation". In: *ACM SIGGRAPH Asia*. 2012.

[151] Sanmit Narvekar et al. "Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey". In: *Journal of Machine Learning Research (JMLR)*. 2020.

[152] OpenAI et al. "Learning Dexterous In-Hand Manipulation". In: *International Journal of Robotics Research (IJRR)*. 2019.

[153] OpenAI et al. "Solving Rubik's Cube with a Robot Hand". In: *arXiv preprint arXiv:1910.07113* (2019).

[154] Takayuki Osa et al. "An Algorithmic Perspective on Imitation Learning". In: *Foundations and Trends in Robotics* 7 (2018).

[155] Fumiaki Osawa, Hiroaki Seki, and Yoshitsugu Kamiya. "Unfolding of Massive Laundry and Classification Types by Dual Manipulator". In: *Journal of Advanced Computational Intelligence and Intelligent Informatics* (2007).

[156] Tom Le Paine et al. "Making Efficient Use of Demonstrations to Solve Hard Exploration Problems". In: *International Conference on Learning Representations (ICLR)*. 2020.

[157] Samuel Paradis et al. "Intermittent Visual Servoing: Efficiently Learning Policies Robust to Instrument Changes for High-precision Surgical Manipulation". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[158] J. K. Parker et al. "Robotic Fabric Handling for Automating Garment Manufacturing". In: *Journal of Manufacturing Science and Engineering* 105 (1983).

[159] Deepak Pathak et al. "Zero-Shot Visual Imitation". In: *International Conference on Learning Representations (ICLR)*. 2018.

[160] Bilal Piot, Matthieu Geist, and Olivier Pietquin. "Boosted bellman residual minimization handling expert demonstrations". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2014, pp. 549–564.

[161] Tobias Pohlen et al. "Observe and Look Further: Achieving Consistent Performance on Atari". In: *arXiv preprint arXiv:1805.11593* (2018).

[162] Dean A Pomerleau. *Alvinn: An Autonomous Land Vehicle in a Neural Network*. Tech. rep. Carnegie-Mellon University, 1989.

[163] Dean A. Pomerleau. "Efficient Training of Artificial Neural Networks for Autonomous Navigation". In: *Neural Comput.* 3 (1991).

[164] Charles Poynton. *Digital Video and HDTV Algorithms and Interfaces*. 1st ed. Morgan Kaufmann Publishers Inc., 2003.

[165] Xavier Provot. "Deformation Constraints in a Mass-Spring Model to Describe Rigid Cloth Behavior". In: *Graphics Interface*. 1995.

[166] Jianing Qian et al. "Cloth Region Segmentation for Robust Grasp Selection". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[167] Arnau Ramisa et al. "Using Depth and Appearance Features for Informed Robot Grasping of Highly Wrinkled Clothes". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2012.

[168] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[169] Ugo Rosolia and Francesco Borrelli. "Learning how to Autonomously Race a Car: a Predictive Control Approach". In: *IEEE Transactions on Control Systems Technology*. 2019.

[170] 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.

[171] Reuven Rubinstein. "The Cross-Entropy Method for Combinatorial and Continuous Optimization". In: *Methodology And Computing In Applied Probability* (1999).

[172] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[173] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Prentice Hall Press, 2009.

[174] Andrei A. Rusu et al. "Policy Distillation". In: *International Conference on Learning Representations (ICLR)*. 2016.

[175] Fereshteh Sadeghi and Sergey Levine. "CAD2RL: Real Single-Image Flight without a Single Real Image". In: *Robotics: Science and Systems (RSS)*. 2017.

[176] Tim Salimans and Richard Chen. "Learning Montezuma's Revenge From a Single Demonstration". In: *arXiv preprint arXiv:1812.03381* (2018).

[177] Jose Sanchez et al. "Robotic Manipulation and Sensing of Deformable Objects in Domestic and Industrial Applications: a Survey". In: *International Journal of Robotics Research (IJRR)*. 2018.

[178] Amrita Sawhney et al. "Playing with Food: Learning Food Item Representations through Interactive Exploration". In: *International Symposium on Experimental Robotics (ISER)*. 2020.

[179] Tom Schaul et al. "Prioritized Experience Replay". In: *International Conference on Learning Representations (ICLR)*. 2016.

[180] Connor Schenck et al. "Learning Robotic Manipulation of Granular Media". In: *Conference on Robot Learning (CoRL)*. 2017.

[181] Simon Schmitt et al. "Kickstarting Deep Reinforcement Learning". In: *arXiv preprint arXiv:1803.03835* (2018).

[182] Johannes Schrimpf and Lars Erik Wetterwald. "Experiments Towards Automated Sewing With a Multi-Robot System". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2012.

[183] John Schulman et al. "Learning from Demonstrations Through the Use of Non-Rigid Registration". In: *International Symposium on Robotics Research (ISRR)*. 2013.

[184] 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.

[185] Daniel Seita et al. "Deep Transfer Learning of Pick Points on Fabric for Robot Bed-Making". In: *International Symposium on Robotics Research (ISRR)*. 2019.

[186] Daniel 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.

[187] Daniel Seita et al. "Learning to Rearrange Deformable Cables, Fabrics, and Bags with Goal-Conditioned Transporter Networks". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[188] Daniel Seita et al. "ZPD Teaching Strategies for Deep Reinforcement Learning from Demonstrations". In: *arXiv preprint arXiv:1910.12154* (2019).

[189] S. Sen et al. "Automating Multiple-Throw Multilateral Surgical Suturing with a Mechanical Needle Guide and Sequential Convex Optimization". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2016.

[190] Kap-Ho Seo et al. "Bed-Type Robotic System for the Bedridden". In: *IEEE International Conference on Advanced Intelligent Mechatronics*. 2005.

[191] Syohei Shibata et al. "A Trajectory Generation of Cloth Object Folding Motion Toward Realization of Housekeeping Robot". In: *International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2012.

[192] Changyeob Shin et al. "Autonomous Tissue Manipulation via Surgical Robot Using Learning Based Model Predictive Control". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[193] Noah Y Siegel et al. "Keep Doing What Worked: Behavioral Modelling Priors for Offline Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2020.

[194] David Silver et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search". In: *Nature* 529 (2016), pp. 484–503. URL: `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`.

[195] Petru Soviany et al. "Curriculum Learning: A Survey". In: *arXiv preprint arXiv:2101.10382* (2021).

[196] Sainbayar Sukhbaatar et al. "Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play". In: *International Conference on Learning Representations (ICLR)*. 2018.

[197] Li Sun et al. "A Heuristic-Based Approach for Flattening Wrinkled Clothes". In: *Towards Autonomous Robotic Systems* (2014).

[198] Li Sun et al. "Accurate Garment Surface Analysis using an Active Stereo Robot Head with Application to Dual-Arm Flattening". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015.

[199] Priya Sundaresan et al. "Learning Rope Manipulation Policies Using Dense Object Descriptors Trained on Synthetic Depth Data". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[200] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *Neural Information Processing Systems (NeurIPS)*. 2014.

[201] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. 2nd. Cambridge, MA, USA: MIT Press, 2018.

[202] Jie Tan et al. "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots". In: *Robotics: Science and Systems (RSS)*. 2018.

[203] A. K. Tanwani et al. "A Fog Robotics Approach to Deep Robot Learning: Application to Object Recognition and Grasp Planning in Surface Decluttering". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2019.

[204] Ian Taylor, Siyuan Dong, and Alberto Rodriguez. "GelSlim3.0: High-Resolution Measurement of Shape, Force and Slip in a Compact Tactile-Sensing Finger". In: *arXiv preprint arXiv:2103.12269* (2021).

[205] Brijen Thananjeyan et al. "Multilateral Surgical Pattern Cutting in 2D Orthotropic Gauze with Deep Reinforcement Learning Policies for Tensioning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[206] 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)*. 2020.

[207] Josh Tobin et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[208] 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)*. 2012.

[209] Eric Torgerson and Fanget Paul. "Vision Guided Robotic Fabric Manipulation for Apparel Manufacturing". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 1987.

[210] Ashish Vaswani et al. "Attention Is All You Need". In: *Neural Information Processing Systems (NeurIPS)*. 2017.

[211] Ashish Vaswani et al. "Tensor2Tensor for Neural Machine Translation". In: *arXiv preprint arXiv:1803.07416* (2018).

[212] Mel Vecerik et al. "A Practical Approach to Insertion with Variable Socket Position Using Deep Reinforcement Learning". In: *arXiv preprint arXiv:1810.01531* (2018).

[213] Mel Vecerik et al. "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards". In: *arXiv preprint arXiv:1707.08817* (2017).

[214] Loup Verlet. "Computer Experiments on Classical Fluids: I. Thermodynamical Properties of Lennard Jones Molecules". In: *Physics Review* 159.98 (1967).

[215] Mickeal Verschoor and Andrei Jalba. "Efficient and Accurate Collision Response for Elastically Deformable Models". In: *ACM Transactions on Graphics* 38 (Mar. 2019), pp. 1–20. DOI: 10.1145/3209887.

[216] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, 1978.

[217] Angelina Wang et al. "Learning Robotic Manipulation through Visual Planning and Acting". In: *Robotics: Science and Systems (RSS)*. 2019.

[218] Zhou Wang et al. "Image Quality Assessment: From Error Visibility to Structural Similarity". In: *Trans. Img. Proc.* (Apr. 2004).

[219] Ziyu Wang et al. "Critic Regularized Regression". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[220] Bryan Willimon, Stan Birchfield, and Ian Walker. "Model for Unfolding Laundry using Interactive Perception". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2011.

[221] Melonee Wise et al. "Fetch & Freight: Standard Platforms for Service Robot Applications". In: *IJCAI Workshop on Autonomous Mobile Service Robots*. 2016.

[222] Alan Wu, AJ Piergiovanni, and Michael Ryoo. "Model-based Behavioral Cloning with Future Image Similarity Learning". In: *Conference on Robot Learning (CoRL)*. 2019.

[223] Jimmy Wu et al. "Spatial Action Maps for Mobile Manipulation". In: *Robotics: Science and Systems (RSS)*. 2020.

[224] Lijun Wu et al. "Learning to Teach with Dynamic Loss Functions". In: *Neural Information Processing Systems (NeurIPS)*. 2018.

[225] Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. "When Do Curricula Work?" In: *International Conference on Learning Representations (ICLR)*. 2021.

[226] Yifan Wu, George Tucker, and Ofir Nachum. "Behavior Regularized Offline Reinforcement Learning". In: *arXiv preprint arXiv:1911.11361* (2019).

[227] Yilin Wu et al. "Learning to Manipulate Deformable Objects without Demonstrations". In: *Robotics: Science and Systems (RSS)*. 2020.

[228] Mengyuan Yan et al. "Self-Supervised Learning of State Estimation for Manipulating Deformable Linear Objects". In: *IEEE Robotics and Automation Letters (RA-L)*. 2020.

[229] Wilson Yan et al. "Learning Predictive Representations for Deformable Objects Using Contrastive Estimation". In: *Conference on Robot Learning (CoRL)*. 2020.

[230] Pin-Chu Yang et al. "Repeatable Folding Task by Humanoid Robot Worker Using Deep Learning". In: *IEEE Robotics and Automation Letters (RA-L)*. 2017.

[231] Michael Yip and Nikhil Das. "Robot Autonomy for Surgery". In: *The Encyclopedia of Medical Robotics* (2017).

[232] Tianhe Yu et al. "COMBO: Conservative Offline Model-Based Policy Optimization". In: *arXiv preprint arXiv:2102.08363* (2021).

[233] Tianhe Yu et al. "MOPO: Model-based Offline Policy Optimization". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[234] Wenzhen Yuan, Siyuan Dong, and Edward H. Adelson. "GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force". In: *Sensors* 17.12 (2017). URL: https://www.mdpi.com/1424-8220/17/12/2762.

[235] Kevin Zakka et al. "Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[236] Andy Zeng. "Learning Visual Affordances for Robotic Manipulation". PhD thesis. Princeton University, 2019.

[237] Andy Zeng et al. "Learning Synergies between Pushing and Grasping with Self-supervised Deep Reinforcement Learning". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.

[238] Andy Zeng et al. "Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[239] Andy Zeng et al. "TossingBot: Learning to Throw Arbitrary Objects with Residual Physics". In: *Robotics: Science and Systems (RSS)*. 2019.

[240] Andy Zeng et al. "Transporter Networks: Rearranging the Visual World for Robotic Manipulation". In: *Conference on Robot Learning (CoRL)*. 2020.

[241] Harry Zhang et al. "Robots of the Lost Arc: Self-Supervised Learning to Dynamically Manipulate Fixed-Endpoint Cables". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2021.

[242] Shangtong Zhang and Richard S. Sutton. "A Deeper Look at Experience Replay". In: *Deep Reinforcement Learning Symposium, NeurIPS*. 2017.

[243] Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. "Automatic Curriculum Learning through Value Disagreement". In: *Neural Information Processing Systems (NeurIPS)*. 2020.

[244] Wenxuan Zhou, Sujay Bajracharya, and David Held. "PLAS: Latent Action Space for Offline Reinforcement Learning". In: *Conference on Robot Learning (CoRL)*. 2020.

Appendix A

Appendix for Chapter 2

A.1 Additional Experiment Information

In addition to using the grasp network to decide on pick points, we also use a *transition network* to decide when the robot has sufficiently managed to smooth one side of the bed. This is a design choice, and we use a separate neural network trained on human labels to reflect how humans may have different preferences regarding the level of "smoothness" of a blanket. The transition network uses the same network architecture and observation, $\mathbf{o}_t$, as the grasp network. It is defined as $\pi_{\theta_T} : \mathbb{R}^{640\times480\times3} \rightarrow \{0, 1\}$ parameterized by θ_T, which maps $\mathbf{o}_t$ to a binary decision $\mathbf{u}_t^{(T)} \in \{0, 1\}$ as to whether the bed frame is sufficiently covered or not. An output of 0 indicates the side closest to the robot is not sufficiently covered, that is, another grasp and pull needs to be attempted by the robot. An output of 1 signifies the side closest to the robot is sufficiently covered. The robot may transition to the other side of the bed. The network was trained with $M = 654$ that were labeled by a human supervisor. The trained network achieved an accuracy of 99% on our validation set.

Figure A.1: Three blankets we use for evaluation. Left: white, used for data collection with a red corner. Center: multicolored yellow and blue (Y&B) with non-homogeneous patterns. Right: teal, with a thin white blanket pinned underneath it.

In this work, we test with three types of blankets, all shown in Figure A.1.

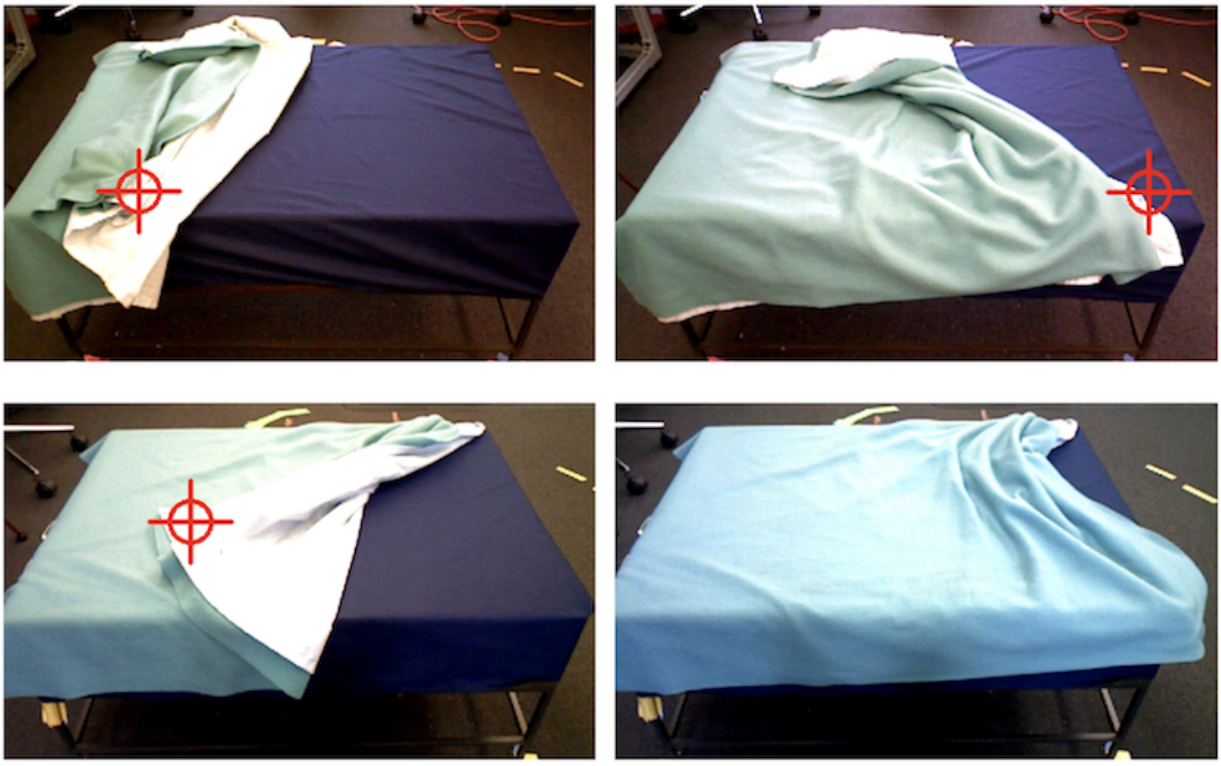

Figure A.2: Two examples of RGB image pairs before and after the robot executes a grasp and pull. Predictions from the grasp network are shown with a red cross hair. Top left: the grasp network suggests a reasonable pick point at the blanket corner, but the robot's grasp and pull did not pull the blanket all the way due to friction, and furthermore, the corner is slightly off the top of the bed frame. In this case (top right) the robot can still make incremental coverage progress. Bottom row: in a different configuration, the grasp network again picks an excellent pick point (left) and this time, one grasp and pull was enough to cover the side (right).

A.2 Additional Qualitative Results

Figure A.2 corresponds to a representative set of blanket configurations seen during task execution under the robot's learned depth-based grasp network. (The figure displays RGB images from the robot's perspective for clarity of presentation, but the robot only used the corresponding depth images for decision-making.)

In the top row of Figure A.2, the grasp network predicted a feasible pick point near a corner, but the robot was unable to pull the blanket all the way to the bed frame corner due to friction and a "weak" grip which had an offset that was slightly too high and thus barely held the corner.[1] In such a situation, the grasp network tends to predict points close

[1]We hand-tuned a gripper height offset, but in rare cases the robot's height offset can be too high. We

to the blanket corner but which are actually top of the bed, given the bias of the training data. The top right image of Figure A.2 shows an example of this reasonable selection; a grasp and pull there allows the robot to make incremental coverage in future attempts.

We note that given the corresponding depth image of the top right image, the transition network π_{θ_T} used in our experiments told the robot to re-attempt a grasp and pull.

In other cases, such as the bottom row of Figure A.2, the grasp network also predicts a pick point near a corner. This time, the robot was able to firmly grip and pull it, and did not need additional grasp attempts as the last image shows a sufficiently stretched blanket.

measure image similarity (using structural similarity [218]) before and after the attempt to check if the robot's gripper was able to affect the blanket.

Appendix B

Appendix for Chapter 3

We structure this appendix as follows:

- Appendix B.1 provides details on the fabric simulator.
- Appendix B.2 discusses the baseline pick point methods from Section 3.4.
- Appendix B.3 explains the imitation learning pipeline, and includes discussion about the neural network architecture and the domain randomization.
- Appendix B.4 describes the experiment setup with the da Vinci Surgical robot.
- Appendix B.5 presents additional experiments with yellow fabric.

B.1 Fabric Environment

The fabric simulator is implemented in Python with Cython for increased speed. The simulator is low fidelity compared to more accurate simulators such as ARCSim [150], but has the advantage of being easier to adapt to the smoothing task we consider and enabling massive data generation. Some relevant hyperparameters are shown in Table B.1.

B.1.1 Actions

The fabric smoothing environment is implemented with a standard OpenAI gym [22] interface. Each action is broken up into six stages: (1) a grasp, (2) a pull up, (3) a pause, (4) a linear pull towards a target, (5) a second pause, and (6) a drop. Steps (2) through (6) involve some number of iterations, where each iteration changes the coordinates of "pinned" points on ξ_t and then calls one "update" method for the fabric simulator to adjust the other, non-pinned points.

Grasp

We implement a grasp by first simulating a gripper moving downwards from a height higher than the highest fabric point, which simulates grasping only the top layer of the fabric. At a given height z_t, to decide which points in ξ_t are grasped given pick point (x_t, y_t), we use a small sphere centered at (x_t, y_t, z_t) with a radius of 0.003 units, where units are scaled so 1 represents the length of a side of the fabric plane. If no points are gripped, then we lower z_t until a point is within the radius. In practice, this means usually 2-5 out of the $25^2 = 625$ points on the cloth are grasped for a given pick point. Once any of the fabric's points are within the gripper's radius, those points are considered fixed, or "pinned".

Pull Up

For 50 iterations, the pull adjusts the z-coordinate of any pinned point by $dz = 0.0025$ units, and keeps their x and y coordinates fixed. In practice, tuning dz is important. If it is too low, an excessive amount of fabric-fabric collisions can happen, but if it is too high, then coverage substantially decreases. In future work, we will consider dynamically adjusting the height change so that it is lower if current fabric coverage is high.

First Pause

For 80 iterations, the simulator keeps the pinned points fixed, and lets the non-pinned points settle.

Linear Pull to Target

To implement the pull, we adjust the x and y coordinates of all pinned points by a small amount each time step (leaving their z coordinates fixed), in accordance with the deltas $(\Delta x_t, \Delta y_t)$ in the action. The simulator updates the position of the non-pinned points based on the implemented physics model. This step is run for a variable amount of iterations based on the pull length.

Second Pause

For 300 iterations, the simulator keeps the pinned points fixed, and lets the non-pinned points settle. This period is longer than the first pause because normally more non-pinned points are moving after a linear pull to the target compared to a pull upwards.

Drop

Finally, the pinned points are "un-pinned" and thus are allowed to lower due to gravity. For 1000 iterations, the simulator lets the entire cloth settle and stabilize for the next action.

Table B.1: Fabric simulator hyperparameters. The spring constant k_s is in Equation 3.1 and damping d is in Equation 3.4.

Hyperparameter	**Value**
Number of Points	$25 \times 25 = 625$
Damping d	0.020
Spring Constant k_s	5000.0
Self-collision thickness	0.020
Height change dz per iteration	0.0025

B.1.2 Starting State Distributions

We provide details on how we generate starting states from the three distributions we use (see Section 3.3.2).

- **Tier 1**. We perform a sequence of two pulls with pick point randomly chosen on the fabric, pull direction randomly chosen, and pull length constrained to be short (about 10-20% of the length of the fabric plane). If coverage remains above 90% after these two pulls we perform a third short (random) pull.
- **Tier 2**. For this tier only, we initialize the fabric in a vertical orientation with tiny noise in the direction perpendicular to the plane of the fabric. Thus the first action in Tier 2 initialization is a vertical drop over one of two edges of the plane (randomly chosen). We then randomly pick one of the two corners at the top of the dropped fabric and drag it approximately toward the center for about half of the length of the plane. Finally we grip a nearby point and drag it over the exposed corner in an attempt to occlude it, again pulling for about half the length of the plane.
- **Tier 3**. Initialization consists of just one high pull. We choose a random pick point, lift it about 4-5 times as high as compared to a normal action, pull in a random direction for 10-25% of the length of the plane, and let the fabric fall, which usually creates less coverage and occluded fabric corners.

The process induces a distribution over starting states, so the agent never sees the same starting fabric state.

These starting state distributions do not generally produce a setting when we have a single corner fold that is visible on top of the fabric, as that case was frequently shown and reasonably approached in prior work [185].

B.2 Details on Baseline Policies

We describe the implementation of the analytic methods from Section 3.4 in more detail.

B.2.1 Highest (Max z)

We implement this method by using the underlying state representation of the fabric ξ_t, and not the images $\mathbf{o}_t$. For each ξ_t, we iterate through all fabric point masses and obtain the one with the highest z-coordinate value. This provides the pick point. For the pull vector, we deduce it from the location of the fabric plane where it would be located if the fabric were perfectly flat.

To avoid potentially getting stuck repeatedly pulling the same point (which happens if the pull length is very short and the fabric ends up "resetting" to the prior state), we select the five highest points on the fabric, and then randomize the one to pick. This policy was able to achieve reasonable coverage performance in Seita et al. [185].

B.2.2 Wrinkles

The implementation approximates the method in Sun et al. [197]; implementing the full algorithm is difficult due to its complex, multi-stage nature and the lack of open source code. Their wrinkle detection method involves computing variance in height in small neighborhoods around each pixel in the observation image $\mathbf{o}_t$, k-means clustering on the pixels with average variance above a certain threshold value, and hierarchical clustering on the clusters found by k-means to obtain the largest wrinkle. We approximate their wrinkle detection method by isolating the point of largest local variance in height using ξ_t. Empirically, this is accurate at selecting the largest wrinkle. To estimate wrinkle direction, we find the neighboring point with the next largest variance in height.

We then pull perpendicular to the wrinkle direction. Whereas Sun et al. [197] constrains the perpendicular angle to be one of eight cardinal directions (north, northeast, east, southeast, south, southwest, west, northwest), we find the exact perpendicular line and its two intersections with the edges of the fabric. We choose the closer of these two as the pick point and pull to the edge of the plane.

B.2.3 Oracle

For the oracle policy, we assume we can query the four corners of the fabric and know their coordinates. Since we know the four corners, we know which of the fabric plane corners (i.e., the targets) to pull to. We pick the pull based on whichever fabric corner is furthest from its target.

B.2.4 Oracle Expose

The oracle expose policy is an extension to the oracle policy. In addition to knowing the exact position of all corners, the policy is also aware of the occlusion state of all corners. The occlusion state is a 4D boolean vector which indicates a 1 if a given corner is visible to the camera from the top-down view or a 0 if it is occluded. The oracle expose policy will try

Table B.2: Hyperparameters for the main DAgger experiments.

Hyperparameter	**Value**
Parallel environments	10
Steps per env, between gradient updates	20
Gradient updates after parallel steps	240
Minibatch size	128
Supervisor (offline) episodes	2000
Policy learning rate	1e-4
Policy L_2 regularization parameter	1e-5
Behavior Cloning epochs	500
DAgger steps after Behavior Cloning	50000

to solve all visible corners in a similar manner to the oracle policy using its extended state knowledge. If all four corners are occluded, or all visible corners are within a threshold of their target positions, the oracle expose policy will perform a revealing action on an occluded corner. We implement the revealing action as a fixed length pull 180 degrees away from the angle to the target position. This process is repeated until the threshold coverage is achieved. We did not use this as the algorithmic supervisor since its performance was not significantly better than the simpler oracle policy.

B.3 Details on Imitation Learning

B.3.1 DAgger Pipeline

Each DAgger "iteration" rolls out 10 parallel environments for 20 steps each (hence, 200 total new samples) which are labeled by the oracle corner policy. These are added to a growing dataset of samples which includes the supervisor's original offline data. After 20 steps per parallel environment, we draw 240 minibatches of size 128 each for training. Then the process repeats with the agent rolling out its new policy. DAgger hyperparameters are in Table B.2. In practice, the L_2 regularization for the policy impacts performance significantly. We use 10^{-5} as using 10^{-4} or 10^{-3} leads to substantially worse performance. We limit the total number of DAgger steps to 50,000 due to compute and time limitations; training for substantially more steps is likely to yield further improvements.

As described in Section 3.6, the policy neural network architecture has four convolutional layers, each with 32 filters of size 3×3, followed by dense layers of size 256 each, for a total of about 3.4 million parameters. The parameters and their dimensions, assuming the input is RGB (with three channels) and 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)
```

Figure B.1: Correcting the dVRK when it slightly misses fabric. Left: the dVRK executes a pick point (indicated with the red circle) but barely misses the fabric. Middle: it detects that the fabric has not changed, and the resulting action is constrained to be closer to the center and touches the fabric (light pink circle). Right: the pull vector results in high coverage.

```
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
```

B.3.2 Domain Randomization and Simulated Images

To transfer a policy to a physical robot, we use domain randomization [207] on color (RGB) and depth (D) images during training. We randomize fabric colors, the shading of the fabric plane, and camera pose. We do not randomize the simulator's parameters.

Figures B.2 and B.3 show examples of simulated images with domain randomization applied. We specifically applied the following randomization, in order:

- For RGB images, we apply color randomization. The fabric background and foreground colors are set at default RGB values of $[0.07, 0.30, 0.90]$ and $[0.07, 0.05, 0.60]$, respectively, creating a default blue color. With domain randomization, we create a random noise vector of size three where each component is independently drawn from Unif$[-0.35, 0.35]$ and then add it to both the background and foreground colors. Empirically, this creates images of various shades "centered" at the default blue value.

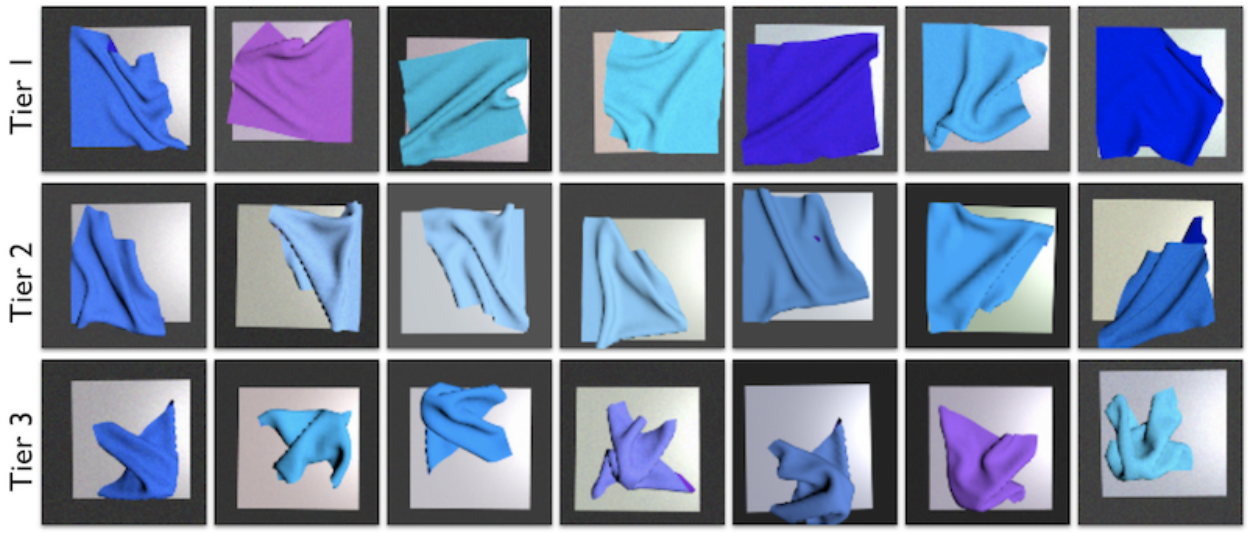

Figure B.2: Representative simulated color images of examples of starting fabric states drawn from the distributions specified in Section 3.3.2. All images are of dimension $100 \times 100 \times 3$. Top row: tier 1. Middle row: tier 2. Bottom row: tier 3. Domain randomization is applied on the fabric color, the shading of the white background plane, the camera pose, and the overall image brightness, and then we apply uniform random noise to each pixel.

- For both RGB and D images, we apply camera pose randomization, with Gaussian noise added independently to the six components of the pose (three for position using meters, three for orientation using degrees). Gaussians are drawn centered at zero with standard deviation 0.04 for positions and 0.9 for degrees.
- After Blender produces the image, we next adjust the brightness of RGB images via OpenCV gamma corrections[1], with $\gamma = 1$ representing no brightness change. We draw $\gamma \sim \text{Unif}[0.8, 1.2]$ for RGB images. To make simulated depth images darker and more closely match the real images, we draw a random value $\beta \sim \text{Unif}[-50, -40]$ and add β to each pixel.

Only after the above are applied, do we then independently add uniform noise to each pixel. For each full image with pixel values between 0 and 255, we draw a uniform random variable $l \sim \text{Unif}[-15, 15]$ between -15 and 15. We then draw additive noise $\epsilon \sim \text{Unif}[-l, l]$ for each pixel independently.

[1] `https://www.pyimagesearch.com/2015/10/05/opencv-gamma-correction/`

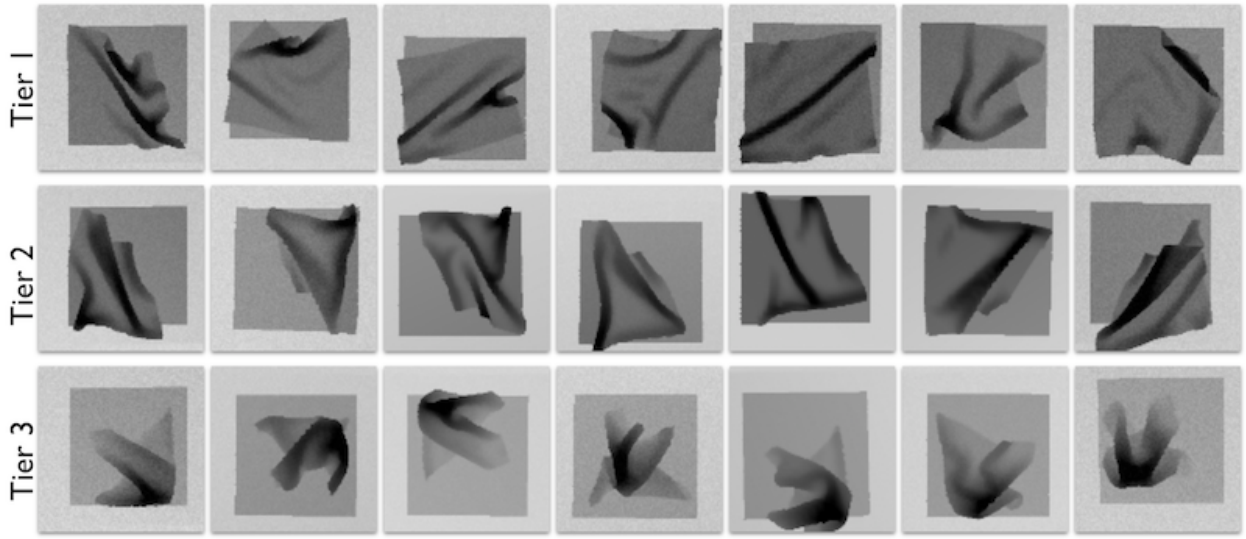

Figure B.3: Representative simulated depth images of examples of starting fabric states drawn from the distributions specified in Section 3.3.2, shown in a similar manner as in Figure B.2. Images are of dimension (100×100). Domain randomization is applied on the camera pose and the image brightness, and then we apply uniform random noise to each pixel.

B.4 Experiment Setup Details

B.4.1 Image Processing Pipeline

The original color and depth images come from the mounted Zivid One Plus RGBD camera, and are processed in the following ordering:

- For depth images only, we apply in-painting to fill in missing values (represented as "NaN"s) in depth images based on surrounding pixel values.
- Color and depth images are then cropped to be 100×100 images that allow the entire fabric plane to be visible, along with some extra background area.
- For depth images only, we clip values to be within a minimum and maximum depth range, tuned to provide depth images that looked reasonably similar to ones processed in simulation. We convert images to three channels by triplicating values across the channels. We then scale pixel values to be within $[0, 255]$ and apply the OpenCV equalize histogram function for all three channels.
- For depth and color images, we apply bilateral filtering and then de-noising, both implemented using OpenCV functions. These help smooth the uneven fabric texture without sacrificing cues from the corners.

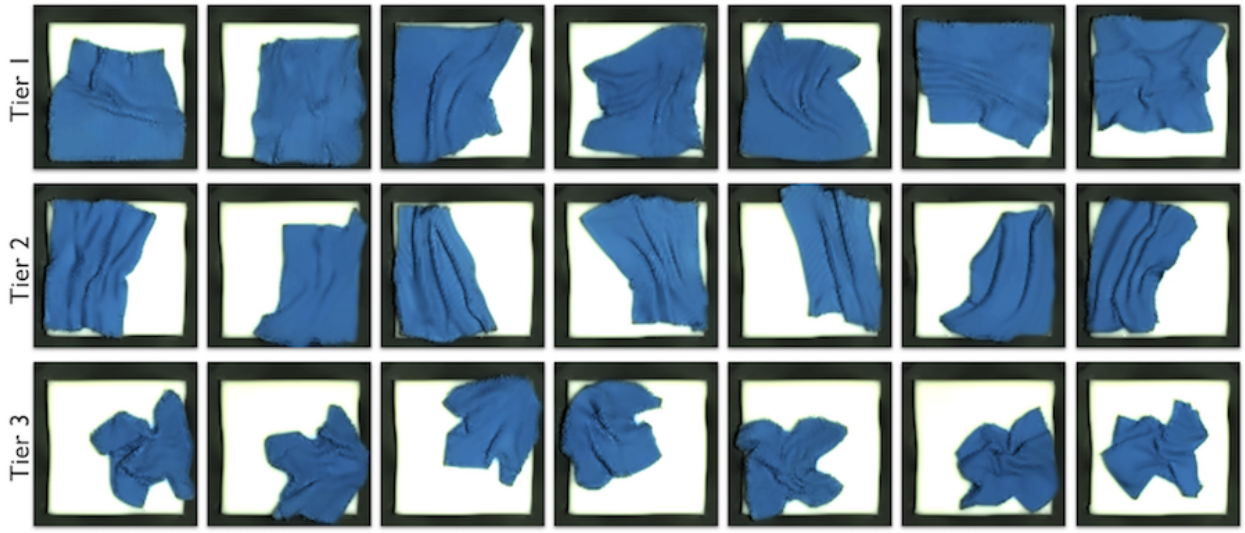

Figure B.4: Representative examples of color images from the physical camera used for the surgical robot experiments, so there is no domain randomization applied here. We manipulated fabrics so that they appeared similar to the simulated states. The color images here correspond to the depth images in Figure B.5.

Figures B.4 and B.5 show examples of real fabric images that policies take as input, after processing. These are then passed as input to the policy neural network.

B.4.2 Physical Experiment Setup and Procedures

To map from neural network output to a position with respect to the robot's frame, we calibrate the positions by using a checkerboard on top of the fabric plane. We move the robot's end effectors with the gripper facing down to each corner of the checkerboard and record positions. During deployment, for a given coordinate frame, we perform bilinear interpolation to figure out the robot position from the four surrounding known points. After calibration, the robot reached positions on the fabric plane to 1-2 mm of error. Figure B.1 shows a visualization of the heuristic we employ to get the robot to grasp fabric when it originally misses by 1-2 mm.

B.5 Additional Experiments

To further test RGB versus D policies, we use the same two policies trained on tier 1 starting states and deploy them on yellow fabric. The color distribution "covered" by domain randomization includes shades of blue, purple, pink, red, and gray, but not yellow. We recreate

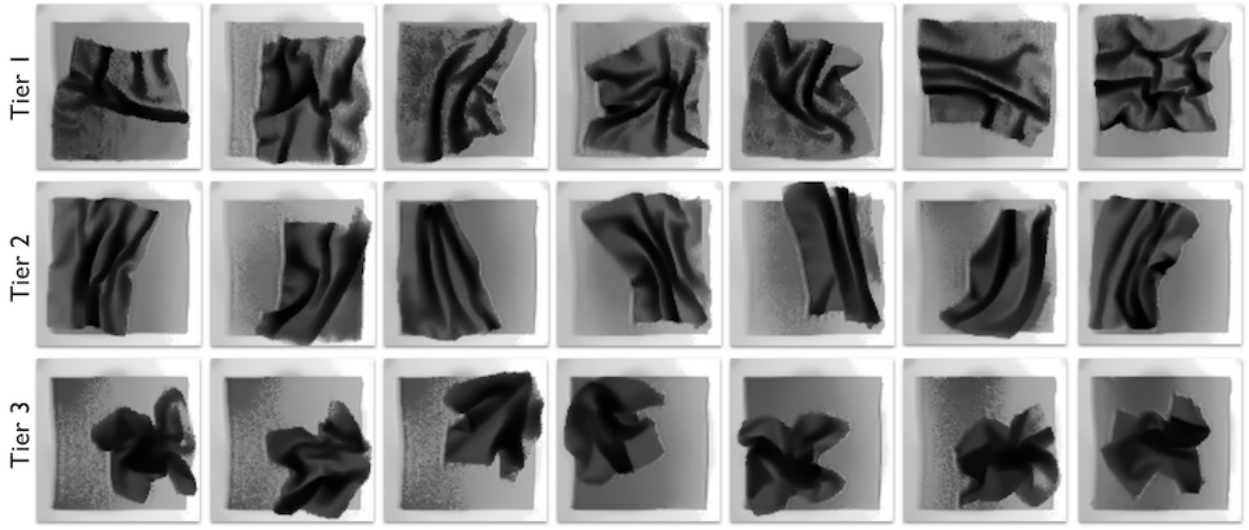

Figure B.5: Representative examples of depth images from the physical camera used for the surgical robot experiments, so there is no domain randomization applied here. We manipulated fabrics so that they appeared similar to the simulated states. The depth images here correspond to the color images in Figure B.4.

Table B.3: Physical experiments for 5 trajectories with yellow fabric. We report the same statistics as in Table 3.2 for the two same policies (T1 RGB and T1 D) trained on tier 1 starting states.

Setting	**(1) Start**	**(2) Final**	**(3) Max**	**(4) Actions**
T1 RGB	81.5 +/- 3	71.7 +/- 25	89.6 +/- 6	7.6 +/- 2
T1 D	83.1 +/- 2	**85.9 +/- 15**	**91.9 +/- 5**	**4.6 +/- 4**

five starting fabric conditions where, with a blue fabric, the RGB policy attained at least 92% coverage in just one action.

Results in Table B.3 indicate poor performance from the RGB policy, as coverage decreases from 81.5% to 71.7%. Only two out of five trajectories result in at least 92% coverage. We observe behavior shown in Figure B.6 where the policy fails to pick at a corner or to pull in the correct direction. The depth-only policy is invariant to colors, and is able to achieve higher ending coverage of 85.9%. This is higher than the 78.8% coverage reported in Table 3.2 due to relatively easier starting states.

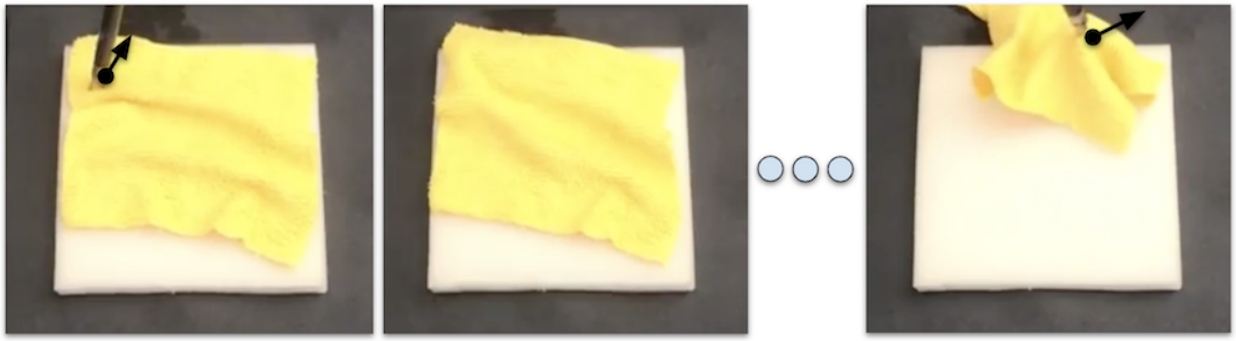

Figure B.6: An example episode taken by a tier 1, RGB-trained policy on yellow fabric. The first action (left image) picks to the upper left and pulls the fabric away from the plane (black arrow). The process repeats for several actions and, as shown in the last image, the fabric barely covers the plane.

Appendix C

Appendix for Chapter 4

We structure this appendix as follows:

- Appendix C.1 compares and contrasts different fabric simulators.
- Appendix C.2 describes, in further detail, the implementation of the learning-based methods (IL, DDPG, VSF) we use for fabric manipulation.
- Appendix C.3 presents additional fabric smoothing results.

C.1 Fabric Simulators

In this work, we use the fabric simulator from [184]. This simulator possesses an ideal balance between ease of code implementation, speed, and accuracy, and was able to lead to reasonable smoothing policies in prior work. We considered using simulators from ARCSim [150], MuJoCo [208], PyBullet [30], or Blender [28], but did not use them for several reasons.

High-fidelity simulators, such as ARCSim, take too long to simulate to get sufficient data for training visual dynamics models. We attempted to simulate rudimentary grasping behavior in ARCSim, but it proved difficult because ARCSim does not represent fabric as a fixed grid of vertices, which meant we could not simulate grasping by "pinning" or "fixing" certain vertices.

The MuJoCo fabric simulator was only recently released in October 2018, and besides concurrent work from Wu et al. [227], at the time of this work, there were no existing environments that combine fabrics with simulated robot grasps. We investigated and used the open-source code from [227], but found that MuJoCo did not accurately simulate fabric-fabric collisions well.

The PyBullet simulator code from Matas et al. [139] showed relatively successful fabric simulation, but it was difficult for us to adapt the author's code to the proposed work, which made significant changes to the off-the-shelf PyBullet code.

Blender includes a new fabric simulator, with substantial improvements after 2017 for more realistic shearing and tensioning. These changes, however, are only supported in Blender 2.8, not Blender 2.79, and we used 2.79 because Blender 2.8 does not allow background processes to run on headless servers, which prevented us from running mass data collection.

Most of these fabric simulators were only recently developed, and some developed concurrently with this work. We will further investigate the feasibility of using these simulators.[1]

C.2 Details of Learning-Based Methods

We describe implementation and training details of the three learning-based methods tested: imitation learning, model-free reinforcement learning, and model-based VisuoSpatial Foresight. The other baselines — random, highest point, and wrinkles — are borrowed unmodified from prior open-source code [184]. To ensure that comparisons are reasonably fair among the methods, we keep hyperparameters as similar as possible.

C.2.1 Imitation Learning Baseline: DAgger

DAgger [170] is implemented directly from the open source DAgger code in Seita et al. [184]. This was originally based on the open-source OpenAI baselines [35] library for parallel environment support to overcome the time bottleneck of fabric simulation.

We ran the corner pulling demonstrator for 2,000 trajectories, resulting in 6,697 image-action pairs $(\mathbf{o}_t, \mathbf{a}'_t)$, where the notation $\mathbf{a}'_t$ indicates the action is labeled and comes from the demonstrator. Each trajectory was randomly drawn from one of the three tiers in the simulator with equal probability. We then perform a behavior cloning [163] "pre-training" period for 200 epochs over this offline data, which does not require environment interaction.

After behavior cloning, each DAgger iteration rolls out 20 parallel environments for 10 steps each (hence, 200 total new samples) which are labeled by the corner pulling policy, the same policy that created the offline data and uses underlying state information. These are inserted into a replay buffer of image-action samples, where all samples have actions labeled by the demonstrator. The replay buffer size is 50,000, but the original demonstrator data of size 6,697 is never removed from it. After environment stepping, we draw 240 minibatches of size 128 each for training and use Adam [101] for optimization. The process repeats with the agent rolling out its updated policy. We run DAgger for 110,000 steps across all environments (hence, 5,500 steps per parallel environment) to make the number of actions consumed to be roughly the same as the number of actions used to train the video prediction model. This is significantly more than the 50,000 DAgger training steps in prior work [184]. Table C.1 contains additional hyperparameters.

[1] After the publication of the paper (Hoque et al. [85]) corresponding to this chapter, Lin et al. [128] released the SoftGym benchmark, which uses NVIDIA FleX simulation and has become a popular choice in the deformable simulation research community due to combining high speed and realistic physics.

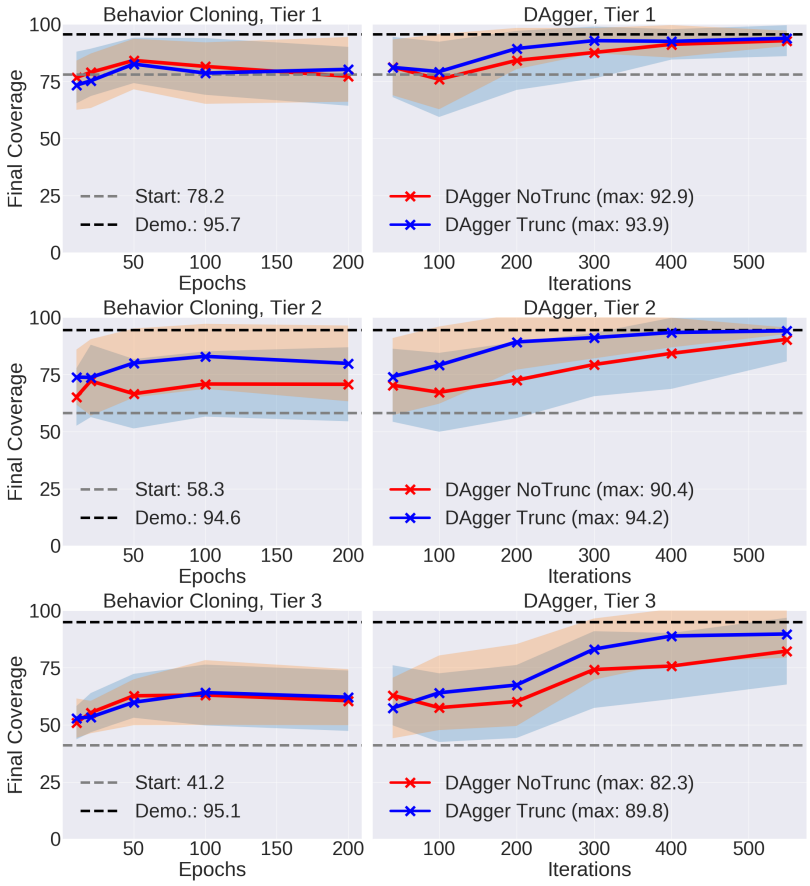

Figure C.1: Average coverage over 50 simulated test-time episodes at checkpoints (marked "X") during the behavior cloning and DAgger phases. For each setting of no action truncation and action truncation, we deploy a single DAgger policy deployed on all tiers. Using dashed lines, we annotate the average starting coverage and the corner pulling demonstrator's average final coverage.

The actor (i.e., policy) neural network for DAgger uses a design based on Seita et al. [184] and Wu et al. [139]. The input to the policy are RGBD images of size $(56 \times 56 \times 4)$, where the four channels are formed from stacking an RGB and a single-channel depth image. The policy processes the input through four convolutional layers that have 32 filters with size 3×3, and then uses four fully connected layers with 256 nodes each. The parameters of the network, ignoring biases for simplicity, are listed as follows:

```
actor/convnet/c1/w   1152 params (3, 3, 4, 32)
actor/convnet/c2/w   9216 params (3, 3, 32, 32)
actor/convnet/c3/w   9216 params (3, 3, 32, 32)
actor/convnet/c4/w   9216 params (3, 3, 32, 32)
actor/fcnet/fc1      663552 params (2592, 256)
actor/fcnet/fc2      65536 params (256, 256)
```

Table C.1: DAgger hyperparameters.

Hyperparameter	**Value**
Parallel environments	20
Steps per env, between gradient updates	10
Gradient updates after parallel steps	240
Minibatch size	128
Discount factor γ	0.99
Demonstrator (offline) samples	6697
Policy learning rate	1e-4
Policy L_2 regularization parameter	1e-5
Behavior Cloning epochs	200
DAgger steps after Behavior Cloning	110000

```
actor/fcnet/fc3      65536 params (256, 256)
actor/fcnet/fc4      1024 params (256, 4)
Total model parameters: 0.83 million
```

The result from the actor policy is a 4D vector representing the action choice $\mathbf{a}_t \in \mathbb{R}^4$ at each time step t. The last layer is a hyperbolic tangent which makes each of the four components of $\mathbf{a}_t$ within $[-1, 1]$. During action truncation, we further limit the two components of $\mathbf{a}_t$ corresponding to the deltas into $[-0.4, 0.4]$.

A set of graphs representing learning progress for DAgger is shown in Figure C.1, where for each marked snapshot, we roll it out in the environment for 50 episodes and measure final coverage. Results suggest the single DAgger policy, when trained with 110,000 total steps on RGBD images, performs well on all three tiers with performance nearly matching the 95-96% coverage of the demonstrator.

We trained two variants of DAgger, one with and one without the action truncation to $[-0.4, 0.4]$ for the two deltas Δx and Δy. The model trained on truncated actions outperforms the alternative setting. Furthermore, it is the setting we use in VisuoSpatial Foresight, hence we use it for physical robot experiments. We choose the final snapshot as it has the highest test-time performance, and we use it as the policy for simulated and real benchmarks in the main part of the chapter.

C.2.2 Model-Free Reinforcement Learning Baseline: DDPG

To provide a second competitive baseline, we apply model-free reinforcement learning. Specifically, we use a variant of Deep Deterministic Policy Gradients (DDPG) [125] with several improvements as proposed in the research literature. Briefly, DDPG is a deep reinforcement learning algorithm which trains parameterized actor and critic models, each of which are normally neural networks. The actor is the policy, and the critic is a value function.

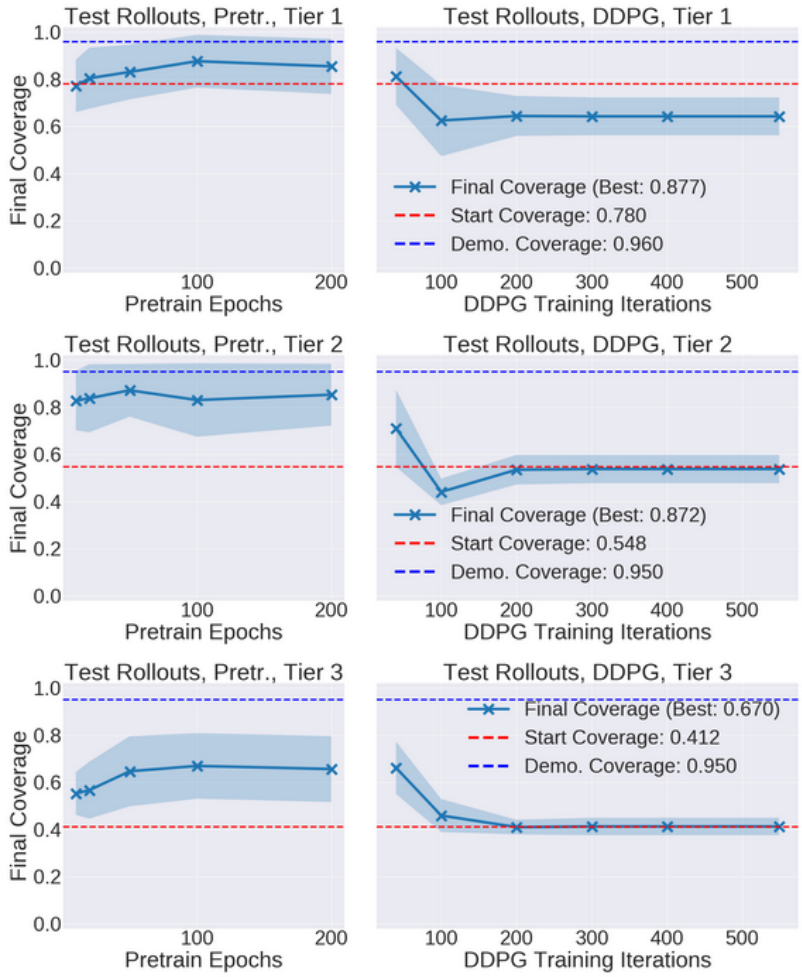

Figure C.2: Average coverage over 50 simulated test-time episodes at checkpoints (marked "X") during the pre-training DDPG phase, and the DDPG phase with agent exploration. This is presented in a similar manner as in Figure C.1 for DAgger. Results suggest that DDPG has difficulty in training a policy that can achieve high coverage.

First, as with DAgger, we use demonstrations [213] to improve the performance of the learned policy. We use the same demonstrator data of 6,697 samples from DAgger, except this time each sample is a tuple of $(\mathbf{o}_t, \mathbf{a}'_t, r_t, \mathbf{o}_{t+1})$, including a scalar reward r_t (to be described) and a successor state $\mathbf{o}_{t+1}$. This data is added to the replay buffer and never removed.

We use a pre-training phase (of 200 epochs) to initialize the actor and critic. We also apply L_2 regularization for both the actor and critic networks. In addition, we use the Q-filter from Nair et al. [148] which may help the actor learn better actions than the demonstrator provides, perhaps for cases when naive corner pulling might not be ideal.

For a fairer comparison, the actor network for DDPG uses the same architecture as the

Table C.2: DDPG hyperparameters.

Hyperparameter	**Value**
Parallel environments	20
Steps per env, between gradient updates	10
Gradient updates after parallel steps	240
Minibatch size	128
Discount factor γ	0.99
Demonstrator (offline) samples	6697
Actor learning rate	1e-4
Actor L_2 regularization parameter	1e-5
Critic learning rate	1e-3
Critic L_2 regularization parameter	1e-5
Pre-training epochs	200
DDPG steps after pre-training	110000

actor for DAgger. The critic has a similar architecture as the actor, with the only change that the action input $\mathbf{a}_t$ is inserted and concatenated with the features of the image $\mathbf{o}_t$ after the four convolutional layers, and before the fully connected portion. As with the imitation learning baseline, the inputs are RGBD images of size $(56 \times 56 \times 4)$.

We design a dense reward to encourage the agent to achieve high coverage. At each time, the agent gets reward based on:

- A small negative living reward of -0.05
- A small negative reward of -0.05 for failing to grasp any point on the fabric (i.e., a wasted grasp attempt).
- A delta in coverage based on the change in coverage from the current state and the prior state.
- A +5 bonus for triggering 92% coverage.
- A -5 penalty for triggering an out-of-bounds condition, where the fabric significantly exceeds the boundaries of the underlying fabric plane.

We designed the reward function by informal tuning and borrowing ideas from the reward in OpenAI et al. [152], which used a delta in joint angles and a similar bonus for moving a block towards a target, or a penalty for dropping it. Intuitively, an agent may learn to take a slightly counter-productive action which would decrease coverage (and thus the delta reward component is negative), but which may enable an easier subsequent action to trigger a high bonus. This reward design is only suited for smoothing. As with the imitation learning baseline, the model-free DDPG baseline is not designed for non-smoothing tasks.

Table C.3: Visual MPC hyperparameters.

Hyperparameter	**Value**
Number of CEM iterations	10
CEM population size	2000
CEM α	0.1
CEM planning horizon	5
CEM initial mean μ	(0, 0, 0, 0)
CEM initial variance Σ	(0.67, 0.67, 0.24, 0.24)

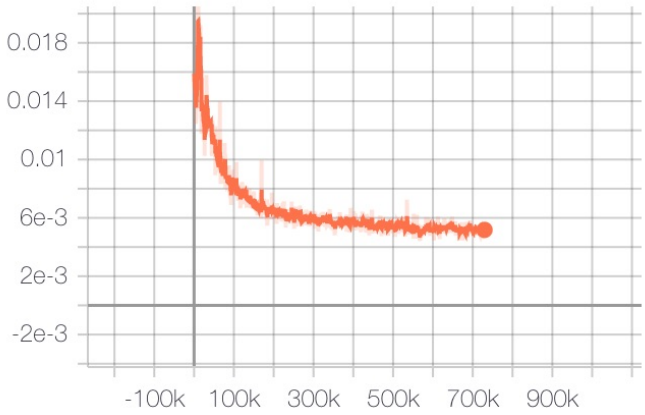

Figure C.3: VSF dynamics model loss curve.

Figure C.2 suggests that the pre-training phase, where the actor and critic are trained on the demonstrator data, helps increase coverage. The DDPG portion of training, however, results in performance collapse to achieving no net coverage. Upon further inspection, this is because the actions collapsed to having no "deltas," so the robot reduces to picking up but then immediately releasing the fabric. Due to the weak performance of DDPG, we do not benchmark the policy on the physical robot.

C.2.3 VisuoSpatial Foresight

The main technique considered in this chapter is VisuoSpatial Foresight (VSF), an extension of Visual Foresight [38]. It consists of a training phase followed by a planning phase. The training phase is as described in Section 4.3.4: we collect 7,003 random episodes with 15 actions and 16 $56 \times 56 \times 4$ RGBD observations each. We use the SV2P [7] implementation in [211]. We set the number of input channels to 4 for RGBD data and predict $H = 7$ output frames from $m = 3$ context frames. During planning, we predict $H = 5$ output frames from $m = 1$ context frame. See Figure C.3 for the loss curve.

Table C.4: Mann-Whitney Test p-values for coverage and number of actions on VSF compared with Random, Highest, Wrinkle and DDPG baselines across all tiers of difficulty.

Tier	**Policy**	**Coverage p-value**	**Actions p-value**
1	Random	0.0000	0.0000
1	Highest	0.0000	0.0002
1	Wrinkle	0.0040	0.0015
1	DDPG	0.0044	0.3670
2	Random	0.0000	0.0000
2	Highest	0.0000	0.0000
2	Wrinkle	0.2323	0.0091
2	DDPG	0.0000	0.0000
3	Random	0.0000	0.0000
3	Highest	0.0000	0.0000
3	Wrinkle	0.0030	0.0199
3	DDPG	0.0000	0.0674

For the planning phase, we tuned the hyperparameters in Table C.3. The variance reported in the table is the diagonal covariance used for folding and double folding. We found that for smoothing, a lower variance (0.25, 0.25, 0.04, 0.04) results in better performance, although it does push the policy toward shorter actions. For the pixel distance cost function, we remove the 7 pixels on each side of the image to get rid of the impact of the dark border, i.e. we turn a 56×56 frame into a 42×42 frame.

C.3 Supplementary Smoothing Results

C.3.1 Statistical Significance Tests

We run the Mann-Whitney U test [137] on the coverage and number of action results reported in Table 4.1 for VSF against all baselines other than Imitation Learning, which we wish to perform similarly to. See Table C.4 for computed p-values. We conclude that we can confidently reject the null hypothesis that the values are drawn from the same distribution for all metrics except Tier 2 coverage for Wrinkle and the Tier 1 and Tier 3 number of actions for DDPG ($p < 0.02$). Note that Tier 3 results are most informative, as it is the most difficult tier.

C.3.2 Domain Randomization Ablation

We run 50 simulated smoothing episodes per tier with a policy trained *without* domain randomization and compare with the 200 episodes from Table 3.1. In the episodes without domain randomization, we keep fabric color, camera angle, background plane shading,

and brightness constant at training and testing time. In the episodes with domain randomization, we randomize these parameters in the training data and test in the same setting as the experiments without domain randomization, which can be interpreted as a random initialization of the domain randomized parameters. In particular, we vary the following:

- Fabric color RGB values uniformly between (0, 0, 128) and (115, 179, 255), centered around blue.
- Background plane color RGB values uniformly between (102, 102, 102) and (153, 153, 153).
- RGB gamma correction with gamma uniformly between 0.7 and 1.3.
- A fixed amount to subtract from the depth image between 40 and 50 to simulate changing the height of the depth camera.
- 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 $\mathcal{N}(0, 90)$, in degrees.
- Random noise at each pixel uniformly between -15 and 15.

From the results in Table C.5, we find that the final coverage values are very similar whether or not we use domain randomization on training data, and we conclude our domain randomization techniques do not have an adverse effect on performance in simulation.

To analyze robustness of the policy to variation in the randomized parameters, we also evaluate the former two policies (trained with and without domain randomization) with randomization in the test environment on Tier 3 starting states. Specifically, we change the color of the fabric in fixed increments from its non-randomized setting (RGB (25, 89, 217)) until performance starts to deteriorate. In Table C.6, we observe that the domain randomized policy maintains high coverage within the training range (RGB (0, 0, 128) to (115, 179, 255)) while the policy without domain randomization suffers as soon as the fabric color is slightly altered.

C.3.3 Action Magnitudes

Figure C.4 shows histograms of the action delta magnitudes, computed as $\sqrt{(\Delta x)^2 + (\Delta y)^2}$, for the physical experiments reported in Section 4.5.2. The histograms strongly suggest that the action magnitudes are smaller for VSF as compared to IL.

Table C.5: Coverage and number of actions for simulated smoothing episodes with and without domain randomization on training data, where the domain randomized results are from Table 4.1.

Tier	Domain Randomized?	Coverage	Actions
1	Yes	92.5 ± 2.5	8.3 ± 4.7
1	No	93.0 ± 3.0	6.9 ± 4.1
2	Yes	90.3 ± 3.8	12.1 ± 3.4
2	No	91.2 ± 9.2	8.7 ± 3.6
3	Yes	89.3 ± 5.9	13.1 ± 2.9
3	No	85.1 ± 12.8	9.9 ± 3.9

Table C.6: Coverage and number of actions for Tier 3 simulated smoothing episodes with and without domain randomization on training data, where we vary fabric color in fixed increments. The default blue color is (26, 89, 217), and the color (128, 191, 115) is slightly outside the domain randomization range. Values for the default setting are repeated from Table C.5 and all other data points are averaged over 20 episodes.

RGB Values	DR?	Coverage	Actions
(26, 89, 217)	Yes	89.3 ± 5.9	13.1 ± 2.9
(51, 115, 191)	Yes	89.3 ± 10.3	11.7 ± 3.5
(77, 140, 166)	Yes	91.4 ± 3.1	11.7 ± 3.1
(102, 165, 140)	Yes	85.6 ± 10.1	13.2 ± 2.7
(128, 191, 115)	Yes	54.7 ± 6.5	10.3 ± 4.0
(26, 89, 217)	No	85.1 ± 12.8	9.9 ± 3.9
(51, 115, 191)	No	60.7 ± 13.6	7.4 ± 2.4

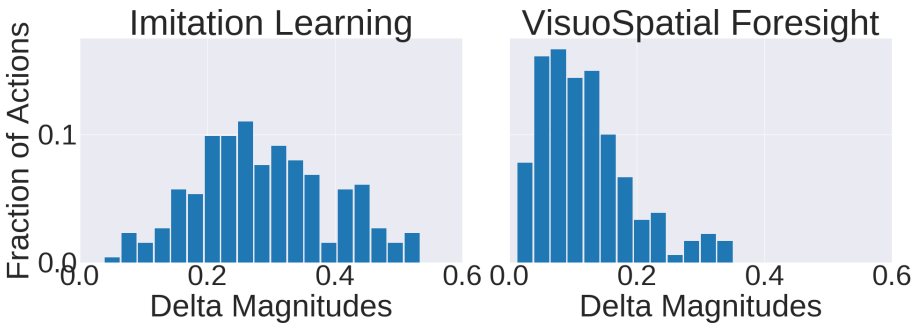

Figure C.4: Histograms showing the distribution of action magnitudes $\sqrt{(\Delta x)^2 + (\Delta y)^2}$ taken by the Imitation Learning and VisuoSpatial Foresight policies from the physical experiments reported in Table 4.3. The y-axis reports each bin as a fraction of the number of actions. The x-axis is consistent among both plots, showing that VSF takes actions with smaller deltas, likely due to the initialization of the mean and variance used to fit the conditional Gaussians for CEM.

Appendix D

Appendix for Chapter 5

This appendix is structured as follows:

- In Appendix D.1 we discuss COVID-19 restrictions, and how results may or may not transfer to the real world.
- In Appendix D.2 we describe the tasks in more detail.
- In Appendix D.3 we define the demonstrator policies.
- In Appendix D.4 we provide more details of Transporter Network architectures.
- In Appendix D.5 we provide details of the experiment methodology.
- In Appendix D.6, we present additional results.

More details, including videos of the learned policies and open-source code, are available on the project website.[1]

D.1 COVID-19 Details

The ongoing COVID-19 pandemic meant that it was infeasible for us to perform physical robotics experiments, which led us to use PyBullet simulation. As in Zeng et al. [240], we strive to avoid system features that would be difficult to replicate in the real world. For example, we assume perception relies on color and depth cameras, which can be established in a physical environment.

One exception is that in simulation, we assume the gripper can "perfectly" grip objects, either a rigid body or a vertex within a soft body (*i.e.,* fabrics and bags). Whenever the simulated suction/pinch gripper comes within a distance threshold to a rigid body or a soft body vertex, it anchors the nearest one to the gripper tip. This is similar to the pinch-grasp

[1] `https://berkeleyautomation.github.io/bags/`

implementations found in [85, 128, 184, 227, 229]. In reality, suction-based grasping or pinch-grasping may be unreliable for certain material properties. To learn policies that capture behaviors robust to such cases, it is ideal to develop environments that explicitly simulate friction and contact forces using these grippers. We leave this investigation to future work.

D.2 Task Details

In *DeformableRavens*, we propose 12 tasks, with details in Table 5.1 and visualizations in Figure 5.2. Here, we elaborate upon the tasks and success criteria in more detail. In addition, we derive the size of the ground truth observation passed into the *GT-State MLP* and *GT-State MLP 2-Step* baseline models (see Section 5.5).

D.2.1 Preliminaries

Each cable is formed from a set of either 24 or 32 rigid beads attached together. Each fabric consists of a grid of 100 vertices, and each bag consists of 353 vertices along with 32 rigid beads to form the bag opening (see Figure 5.4). The ground truth baselines account for every rigid object in a task by collecting the 2D coordinate position and a rotational angle about the z-axis, to get a single pose (x, y, θ). We account for fabrics by taking its 100 vertices and stacking all the 3D vertex positions together, to get a 300D vector. For bags, to reduce the ground truth state complexity and dimensionality, the baselines use ground truth pose information for the 32 rigid beads that form the bag opening, and do not use vertex information. The ground truth state also uses an additional 3D pose representing the displacement vector from the center of the bag's initial pose. Hence, a ground truth description of one bag is a vector of size $33 \times 3 = 99$. This reduced state representation to simplify the input to methods using ground truth information has been used in independent and similar work in benchmarking deformable object manipulation [128].

The four bag tasks have different "task stages," where the pull height and end-effector velocity vary based on the stage. All bag tasks have an initial stage where the robot opens up the bag if necessary. This stage has a low pull height and a slow gripper speed to increase physics stability. Then, *bag-items-1*, *bag-items-2*, and *bag-color-goal* have stages where the robot must insert an item into the bag, and *bag-items-1* and *bag-items-2* additionally have a third stage after that where the robot must transport the bag to a target zone. In these later stages, the pull vector must be high enough so that either a rigid item can be successfully inserted into the bag opening, or that the bag can be lifted in midair. For test-time deployment, these task stages are determined based on a segmented image which determines the object corresponding to a pick point.

D.2.2 DeformableRavens Task Details

(a) cable-ring has a cable with 32 beads attached together in a ring. The robot must open the cable towards a target zone specified by a ring, which contains the same area as the cable with maximized convex hull area. The termination condition is based on whether the area of the convex hull of the cable is above a threshold. Size of ground truth state: with 32 beads and 32 targets (one per bead), each consisting of a 3D description, the state vector is of size $(32 + 32) \times 3 = 192$.

(b) cable-ring-notarget is the same as *cable-ring*, except the target zone is not visible. Size of ground truth state: with 32 beads, the state vector is of size $32 \times 3 = 96$.

(c) cable-shape has a cable with 24 beads attached together, with both endpoints free (*i.e.,* it is not a ring-like cable). The robot must adjust the cable towards a green target shape anchored on the workspace, specified by a series of 2, 3, or 4 line segments attached to each other. Success is based on the fraction of beads that end up within a distance threshold of any point on the target zone. Size of ground truth state: with 24 beads, and 24 targets (one per bead), plus one more pose to represent the center of the target shape, all of which are represented with 3D poses, the state vector is of size $(24 + 24 + 1) \times 3 = 147$.

(d) cable-shape-notarget is the same as *cable-shape*, except the target zone is not visible on the workspace, but specified by a separate goal image of the beads in a desired configuration. In this case, we save the poses of the 24 beads corresponding to the target image, and stack that with the ground-truth state vector. Size of ground truth: with 24 beads for the current cable and 24 for the cable in the target image, the state vector is of size $(24 + 24) \times 3 = 144$.

(e) cable-line-notarget is a simpler version of *cable-shape-notarget*, where the target image is specified with beads that form roughly a straight line. Size of ground truth state: as with *cable-shape-notarget*, the state vector is of size $(24 + 24) \times 3 = 144$.

(f) fabric-cover starts with a flat fabric and a cube. The robot must place the cube onto the fabric, and then fold the fabric so that it covers the cube. Success is based on if the cube is not visible via a top-down object mask segmentation. Size of ground truth state: we use a 300D representation of fabrics, along with 3D pose information from the cube, resulting in a state vector of size $300 + 3 = 303$.

(g) fabric-flat inspired by Seita et al. [184, 185] and Wu et al. [227], the robot must manipulate a square fabric to move it towards a visible target zone of the same size as a flat fabric. The fabric starts flat, but does not fully cover the target zone.[2] Success is based on if the robot exceeds a coverage threshold of 85%, computed via top-down image segmentation. Size of ground truth state: we use a 300D representation of fabrics, along with a 3D pose for the target zone, so the state vector is of size $300 + 3 = 303$.

(h) fabric-flat-notarget is the same as *fabric-flat*, except the target is specified with a separate image showing the fabric in the desired configuration. For ground truth information, we store the 300D representation of the fabric in the target zone, and stack that as input

[2]We tried perturbing the starting fabric configuration to induce folds, but found that it was difficult to tune self-collisions among fabric vertices.

with the current fabric. Size of ground truth state: with 300D representations for the current and target fabrics, the state vector is of size $300 + 300 = 600$.

(i) bag-alone-open has a single bag with random forces applied to perturb the starting configuration. The objective is to expand the bag opening, which is measured via the convex hull area of the ring of beads that form the bag opening. Success is based on whether the convex hull area exceeds a threshold. Size of ground truth state: as reviewed in Appendix D.2.1, each bag is a 99D vector, and that is the size of the state dimension as there are no other items.

(j) bag-items-1 contains one bag, one cube, and one target zone, with forces applied to perturb the initial bag state. The goal is to first open the bag (if necessary), insert the cube into the bag, then lift and transport the bag to the target zone. The task is successful only if the cube is entirely contained in the target zone, and part of the bag opening is also in the target zone. Size of ground truth state: as with *bag-alone-open*, the bag is 99D, and we additionally consider the 3D cube pose and the 3D target zone pose, so the state dimension is $99 + 3 + 3 = 105$.

(k) bag-items-2 is a harder variant of *bag-items-1*, where there exists two items that must be transported to the target zone, and where the two items can be of various shapes and colors. The sizes of the items are constrained so that they can be feasibly inserted within the bag. Size of ground truth state: it contains similar information as *bag-items-1*, but has one more rigid item, so the state dimension is $99 + 3 + 3 + 3 = 108$.

(l) bag-color-goal is a goal conditioned bag task with two bags and one rigid item. The two bags have the same size but different colors. The target image specifies two bags with one item inside it, and the objective is for the robot to put the item in the correct colored bag. Critically, the current state will have a bag of the same color as the target bag, and in the same spatial location. Size of ground truth state: each bag is 99D, and there is one 3D rigid item pose. Finally, each bag is augmented with a 3D vector specifying the RGB color value. The dimension for the *current* state is $(99 + 99 + 3 + 3 + 3) = 207$. This is repeated for the *goal* state, hence the final state dimension is $207 \times 2 = 414$.

D.2.3 Additional Task: Block-Notarget

In addition to tasks in *DeformableRavens*, we create **block-notarget**, for fast prototyping of goal-conditioned Transporter Network architectures. This task is based on the insertion task from Zeng et al. [240], where the robot must pick, rotate, and precisely place an L-shaped block into a target receptacle. To adapt it to the goal-conditioned setting, we remove the the visible receptacle, and instead pass a separate goal image $\mathbf{o}_g$ which contains the receptacle in the desired location. See Figure D.1 for a visual. The ground truth state vector is of size 6, due to having 3D pose information for the current block and target block.

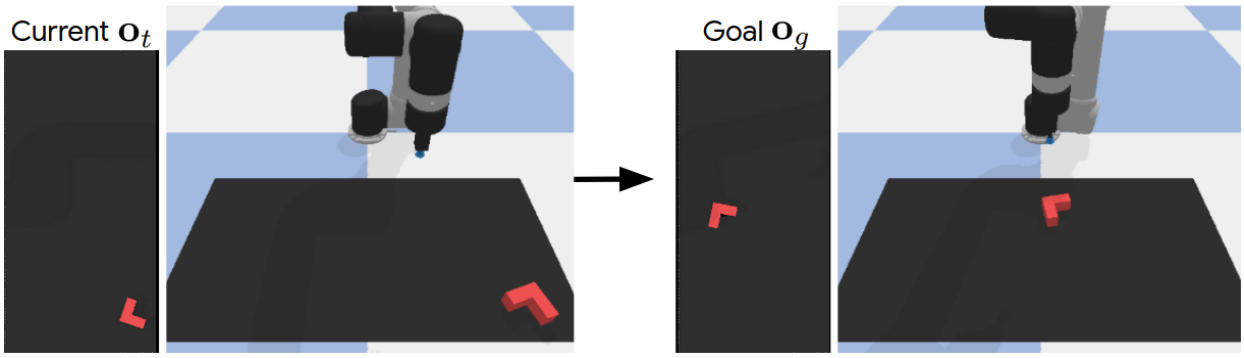

Figure D.1: **block-notarget**. This task involves a red, rigid L-shaped block. At the current observation $\mathbf{o}_t$, the block starts at some random location on the workspace, and the robot must use the same pick and place action formulation to get the block to a target pose specified in the goal image $\mathbf{o}_g$.

D.3 Demonstrator Data Policy

All tasks come with a scripted demonstrator. We briefly highlight how the demonstrator is scripted, and refer the reader to the open-source code for details.

For *cable-shape*, *cable-shape-notarget*, and *cable-line-notarget*, the demonstrator computes, for each cable bead, its distance to the nearest point on the target zone. It performs a pick and place to correct the largest discrepancy. For *fabric-cover*, the demonstrator places the cube onto the center of the flat fabric, then picks a corner at random and pulls it to the opposite corner of the fabric to cover the item. For *fabric-flat* and *fabric-flat-notarget*, we use a corner-pulling demonstrator as in [184] by sequentially correcting for the fabric corner furthest from the target zone corner.

In *cable-ring*, *cable-ring-notarget*, and all four bag tasks, the demonstrator begins with a common sub-procedure. These tasks contain a set of attached beads that either form a ring or the bag's opening (for bag tasks). For each action in this stage, the demonstrator first computes a set of target positions on the workspace (forming a circle) corresponding to what a maximally spread out cable ring would look like. It then enumerates all valid bipartite graphs between the set of cable beads and the set of targets, and picks the one with minimum distance, and concludes with a pick and place to correct the largest bead-to-target discrepancy.

Once the area of the convex hull contained by the beads exceeds a tuned threshold, the task either ends (if *cable-ring*, *cable-ring-notarget*, or *bag-alone-open*) or the demonstrator proceeds to inserting items (if *bag-items-1*, *bag-items-2*, or *bag-color-goal*). For inserting items, the demonstrator picks at a valid item and then places it at a random spot within the bag opening, computed via OpenCV contour detection. Finally, for the bag moving stage in *bag-items-1* and *bag-items-2*, the demonstrator does a pick on a random visible bag bead,

Table D.1: Further details on the scripted demonstrator data performance on tasks in *DeformableRavens*, along with the extra *block-notarget* task. See Table 5.1 in the main text. For each task and its 1000 demonstrator data episodes (which are filtered to only contain successes in the four bag tasks), we report the mean and median length. See Appendix D.3 for details.

Task (Max Ep. Length)	Mean	Median
cable-ring (20)	6.56 ± 3.8	5.0
cable-ring-notarget (20)	6.49 ± 3.7	5.0
cable-shape (20)	6.86 ± 3.3	6.0
cable-shape-notarget (20)	6.94 ± 3.4	6.0
cable-line-notarget (20)	5.55 ± 2.2	6.0
fabric-cover (2)	2.00 ± 0.0	2.0
fabric-flat (10)	3.10 ± 1.5	3.0
fabric-flat-notarget (10)	3.12 ± 1.6	3.0
bag-alone-open (10)	5.63 ± 3.4	5.0
bag-items-1 (8)	2.84 ± 0.8	3.0
bag-items-2 (9)	4.07 ± 0.9	4.0
bag-color-goal (8)	2.60 ± 1.3	2.0
block-notarget (2)	1.00 ± 0.0	1.0

pulls with a hard-coded higher height delta, and ultimately places the bag at the center of the target zone.

We briefly highlight two aspects of the demonstrator:

1. Since the demonstrator has access to ground truth state information (*i.e.,* rigid object poses and deformable vertices), it performs the same on tasks that only differ based on the existence of a visible target zone on the workspace (*e.g., fabric-flat* versus *fabric-flat-notarget*).

2. The demonstrator is stochastic (*i.e.,* behaviors are different between demonstrations). PyBullet provides built-in image segmentation code to find all pixels corresponding to an object. When the demonstrator picks an object, it samples uniformly at random among all pixels corresponding to the target object in the segmentation mask. The stochastic nature, along with randomly sampled object poses at the start of each new episode, ensures some level of dataset diversity.

Since the models we test use behavior cloning, the performance of the learned agents is bounded by the data-generating policy. For the bag tasks, it is difficult to script a high-performing demonstrator given the complexity of bag simulation and manipulation. Therefore, we ignore unsuccessful episodes and run the demonstrator to generate as much data as needed until it gets 1000 successful demos. Due to stochasticity, we hypothesize that seeing

only the successful demonstrations may be sufficient for policies to learn reasonable pick and place patterns from data. For *bag-alone-open*, *bag-items-1*, *bag-items-2*, and *bag-color-goal*, getting 1000 successful demonstrator episodes required running 1661, 2396, 3276, and 1110 times, respectively.

D.4 Additional Transporter Network Details

For standard Transporter Networks [240] (reviewed in Section 5.2.2), we swap the cropping order within the query network Φ_{query}, so that cropping happens after the observation image $\mathbf{o}_t$ passes through the network. This enables subsequent features to use a larger receptive field.

For *Transport-Goal-Split*, we demonstrate the forward pass for the transport module using Tensorflow-style [138] pseudo-code. This module includes Φ_{key}, Φ_{query}, and Φ_{goal}, but does not include the attention module f_{pick}, which is used prior to the transport module forward pass.

```
import numpy as np
import tensorflow as tf
import tensorflow_addons as tfa

# Args:
#   p (pick point tuple, from `f_pick` net)
#   crop_size (64 in our experiments)
#   n_rots (usually 1 in our experiments)
#   input_data and goal_data

# Define FCNs for Transport module.
Phi_key   = FCN()
Phi_query = FCN()
Phi_goal  = FCN()

# input and goal images --> TF tensor.
in_tensor = tf.convert_to_tensor(input_data)
g_tensor  = tf.convert_to_tensor(goal_data)

# Get SE2 rotation vectors for cropping, centered
# at pick point. Assumes it's defined somewhere.
rvecs = get_se2(n_rots, p)

# Forward pass through three separate FCNs.
k_logits = Phi_key(in_tensor)
q_logits = Phi_query(in_tensor)
g_logits = Phi_goal(g_tensor)

# Combine.
g_x_k_logits = tf.multiply(g_logits, k_logits)
g_x_q_logits = tf.multiply(g_logits, q_logits)
```

Table D.2: Some relevant hyperparameters of models tested and reported in Table 5.2. For brevity, we use "Transporters" here to collectively refer to Transporter, Transporter-Goal-Split, and Transporter-Goal-Stack. See Sections 5.2 and 5.3 for background on notation for Transporter-based models. We also report values for the ground truth (GT) baseline models.

Hyperparameter	Value
Adam Learning Rate ($f_{\text{pick}}, \Phi_{\text{key}}, \Phi_{\text{query}}, \Phi_{\text{goal}}$)	1e-4
Adam Learning Rate (GT)	2e-4
Batch Size (Transporters)	1
Batch Size (GT)	128
Training Iterations (Transporters and GT)	20,000

```
# Crop the kernel_logits about the picking point,
# with the help of TensorFlow's image transforms.
crop = tf.identity(g_x_q_logits)
crop = tf.repeat(crop, repeats=n_rots, axis=0)
crop = tfa.image.transform(crop, rvecs)
kernel = crop[:,
              p[0]:(p[0] + crop_size),
              p[1]:(p[1] + crop_size),
              :]

# Cross-convolve!
output = tf.nn.convolution(g_x_k_logits, kernel)
return (1 / (crop_size**2)) * output
```

D.5 Additional Experiment Details

Zeng et al. [240] test with several baseline models in addition to Transporters: Form2Fit [235], a CNN followed by fully connected layers for actions [119], two ground-truth state baselines, and ablations to Transporter Networks. In an effort to alleviate compute, we only use the best versions of the Transporter Networks and Ground-Truth methods in [240], and we do not use the Form2Fit or CNN baselines, which exhibited significantly worse performance than Transporters. We show relevant hyperparameters, such as the Adam [101] learning rate, in Table D.2.

As described in Section 5.6, for each model type and demo count $N \in \{1, 10, 100, 1000\}$, we train for 20K iterations, where each iteration consists of one gradient update from one minibatch of data. We save snapshots every 2K iterations, resulting in 10 snapshots. Each snapshot is then loaded to run 20 episodes using held-out "test set" random seeds. We repeat this process by considering 3 separate training runs (per model type and demo count) to get more evaluation data points. The exception is with *bag-color-goal*, where we train and

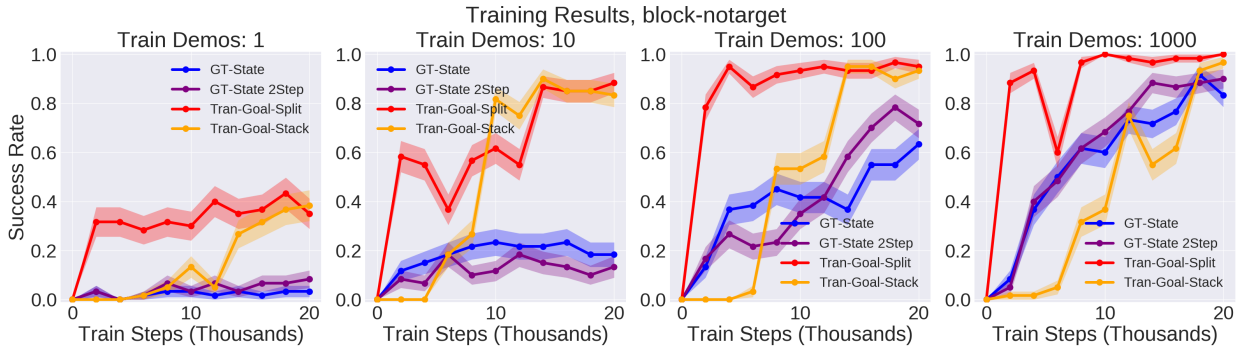

Figure D.2: Results for models on the *block-notarget* task. From left to right, we report models trained with 1, 10, 100, and 1000 demonstrations. All models are trained for 20K iterations, and perform test-time evaluation on iterations: 2K, 4K, 6K, 8K, 10K, 12K, 14K, 16K, 18K, 20K. Each data point represents the average of 60 test-time episode evaluations; we train 3 separate training runs, and run each of those snapshots for 20 test-time episodes at each iteration checkpoint. We also report performance of policies at "iteration 0" with random parameters. Shaded regions indicate one standard error of the mean.

Table D.3: **Baseline Comparisons on Block Task**. Task success rate specifically for *block-notarget*, described in Appendix D.2.3 and Figure D.1. The table is formatted in a similar way as in Table 5.2.

	block-notarget			
Method	1	10	100	1000
GT-State MLP	3.3	23.3	63.3	91.7
GT-State MLP 2-Step	8.3	18.3	78.3	90.0
Transporter-Goal-Stack	38.3	90.0	95.0	96.7
Transporter-Goal-Split	43.3	88.3	96.7	100.0

load (*i.e.*, test) just 1 time (instead of 3) for each demo count, due to the computationally intensive nature of testing, which requires simulating environment steps with two bags. For the 100 and 1000 demonstration *bag-color-goal* datasets, we additionally train *Transporter-Goal-Split* and *Transporter-Goal-Stack* for 40K iterations instead of 20K (while keeping the batch size at 1), since preliminary results showed that performance of the models notably improved at around 20K iterations.

D.6 Additional Experiment Results

We find in Table D.3 that *Transporter-Goal-Stack* and *Transporter-Goal-Split* are roughly two orders of magnitude more sample efficient on *block-notarget*, with success rates from 10

demos (88.3% and 90.0% for the two variants) on par with success rates from 1000 demos for the ground truth models (90.0% and 91.7%). Figure D.2 contain learning curves showing the performance of the models as a function of training iterations.

In Figures D.3, D.4, D.5, and D.6, we show similar learning curves for the tasks in *DeformableRavens* (see Table 5.1), where model architectures in each figure are all trained on the same set of 1, 10, 100, and 1000 demonstrations, respectively. Besides the number of demonstrations used for training each of the models, we keep experimental settings consistent across Figures D.3, D.4, D.5, and D.6.

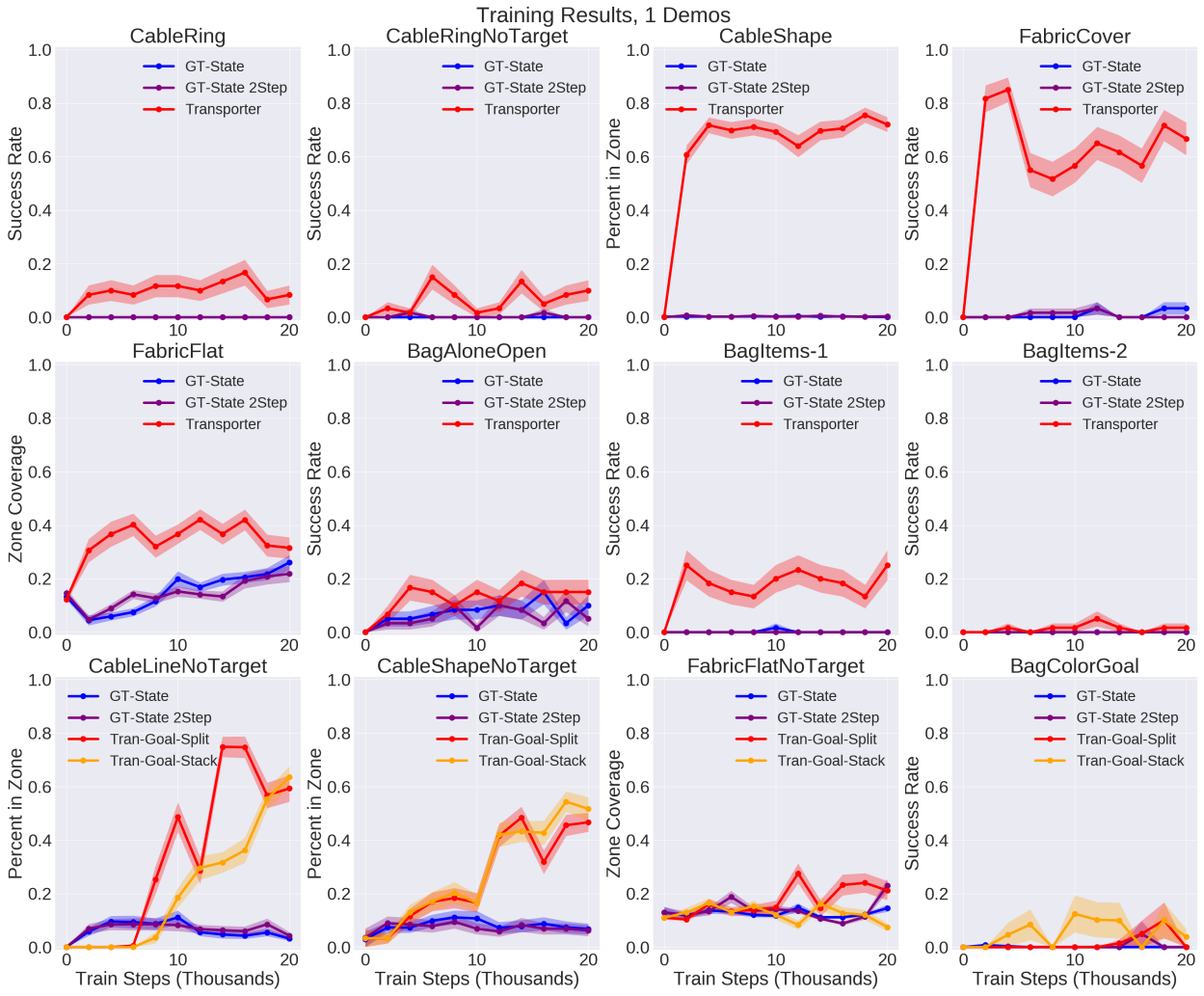

Figure D.3: Results for various models, each trained on **1 demonstration**, with shaded regions indicating one standard error of the mean. We train for 20K iterations, and perform test-time evaluation on iterations: 2K, 4K, 6K, 8K, 10K, 12K, 14K, 16K, 18K, 20K. Each data point represents the average of 60 test-time episode evaluations; we train three separate training runs, and run each snapshot within those runs for 20 test-time episodes. Random seeds for test-time episodes are kept fixed among all methods so that evaluation is done on consistent starting states. We also report performance of policies at "iteration 0" with random parameters.

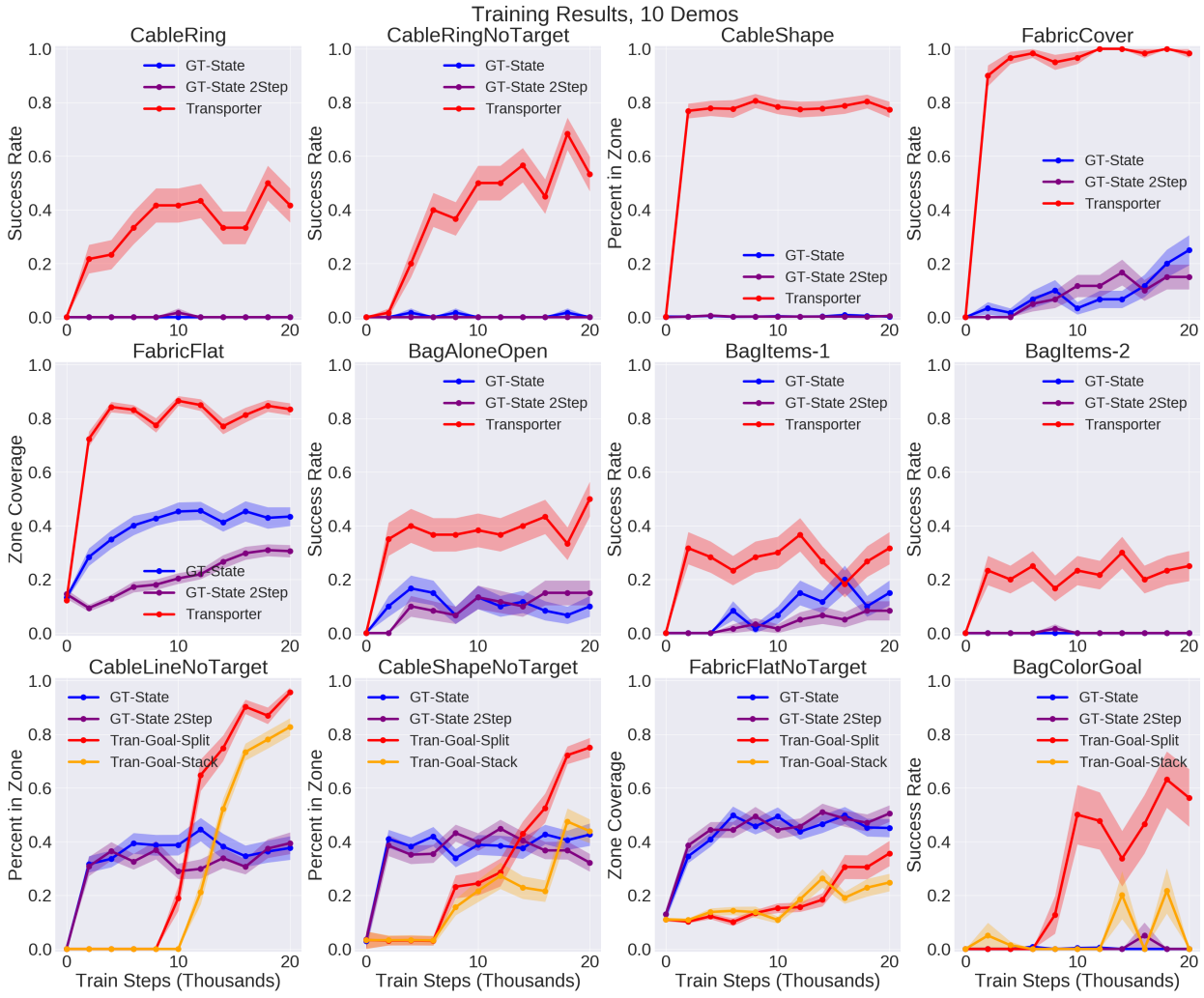

Figure D.4: Results for various models, each trained on **10 demonstrations**. The results and plots are formatted following Figure D.3.

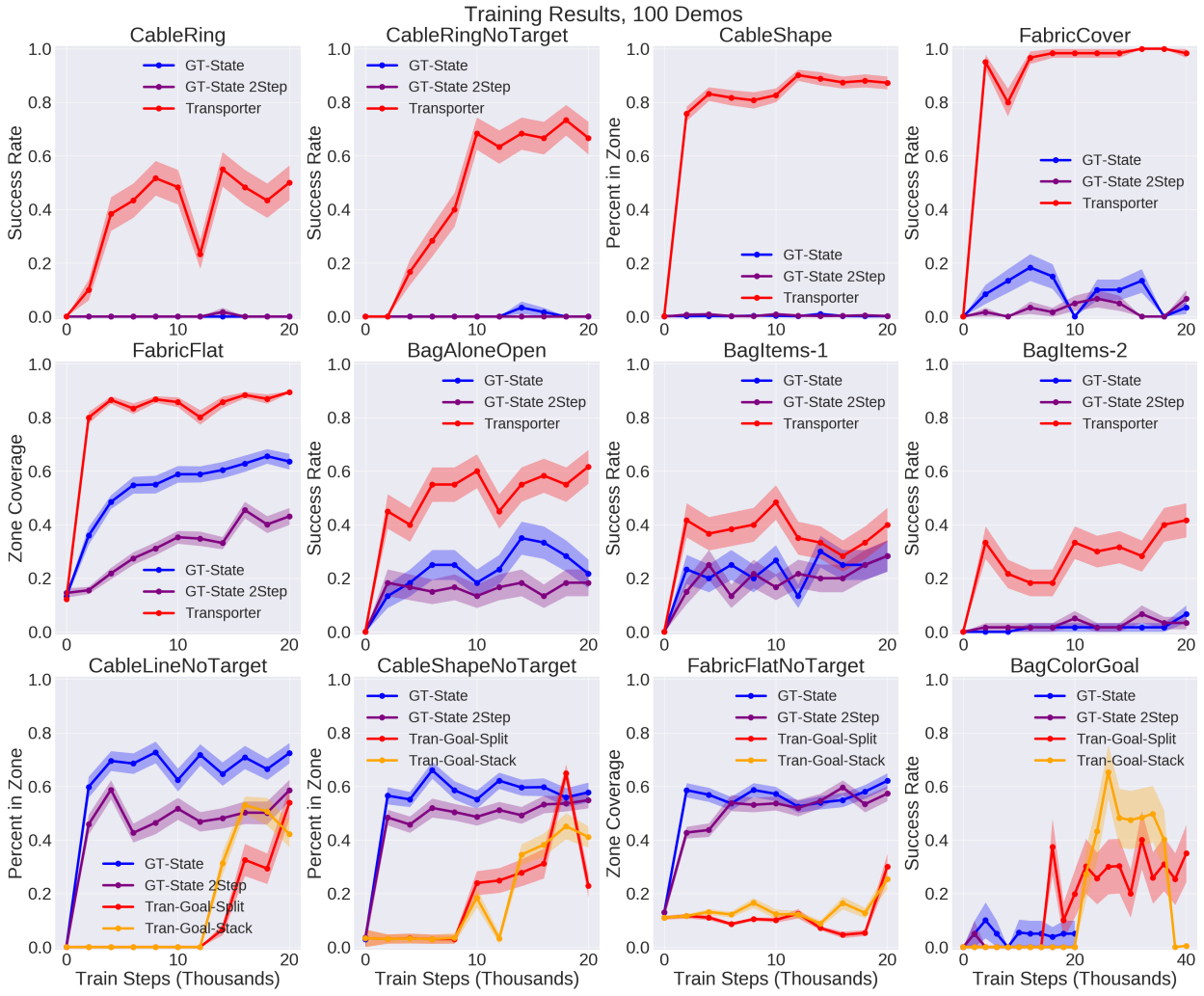

Figure D.5: Results for various models, each trained on **100 demonstrations**. The results and plots are formatted following Figure D.3, with the exception that *Transporter-Goal-Split* and *Transporter-Goal-Stack* are trained with 40K iterations (rather than 20K) for *bag-color-goal*, due to the need to train longer for this task. (The batch size is still 1, meaning that the models utilize fewer samples than ground-truth models that use a batch size of 128.)

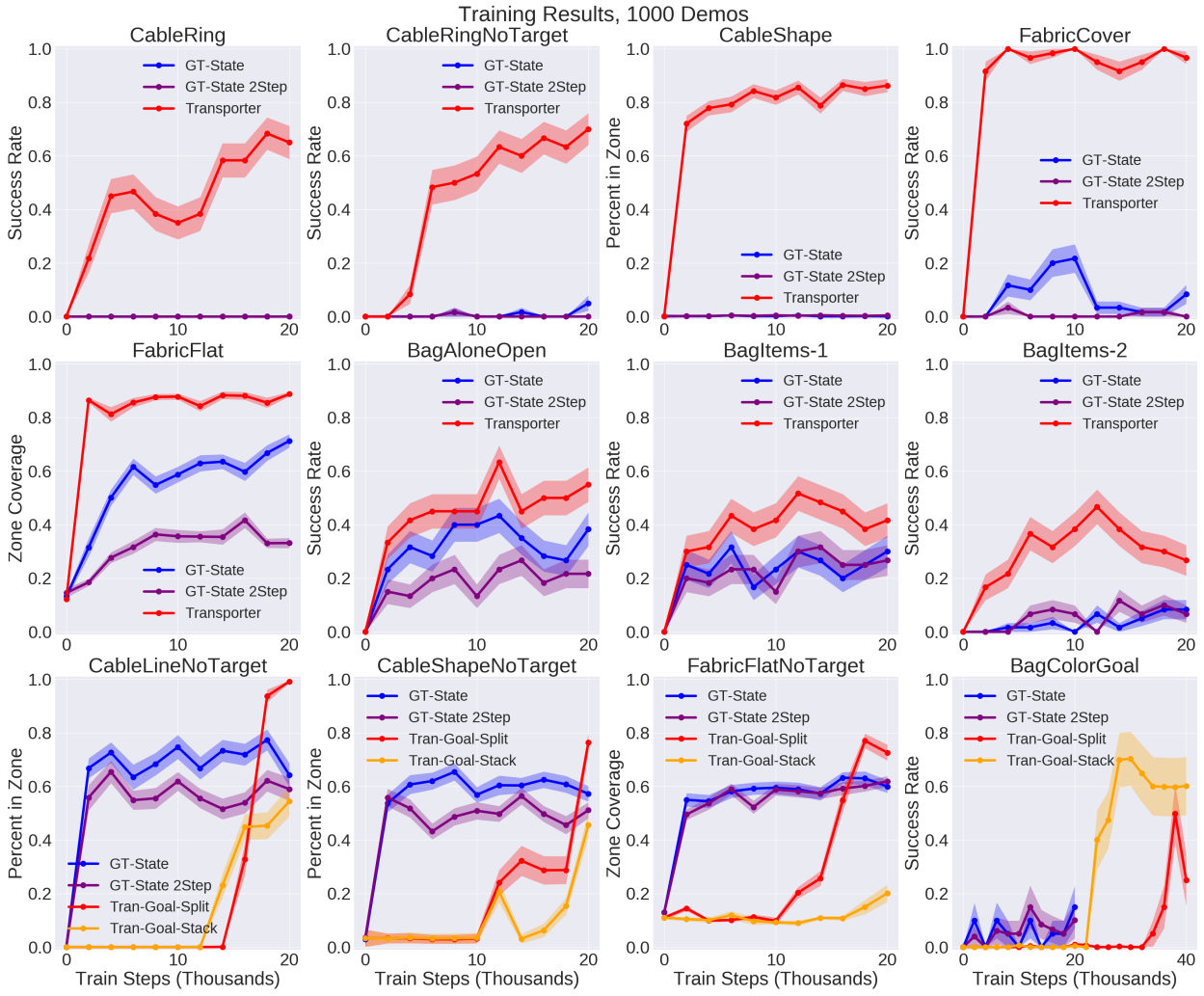

Figure D.6: Results for various models, each trained on **1000 demonstrations**. The results and plots are formatted following Figure D.5.

Appendix E

Appendix for Chapter 7

In this supplementary material, we

- discuss the environments and teacher policy performance (Section E.1),
- describe additional experiment details (Section E.2),
- present additional results (Section E.3).

Additional supplementary material, including code and data, is available on the project website: `https://sites.google.com/view/teach-curr/home`.

E.1 Environments and Teacher Performance

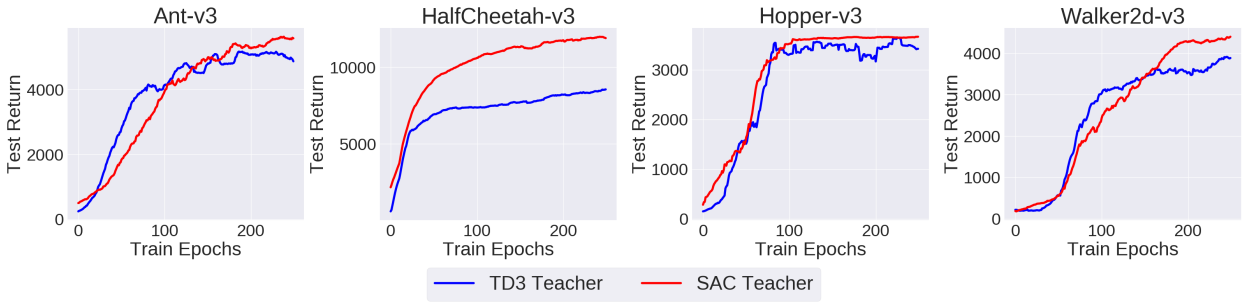

Figure E.1: Teacher performance of TD3 (blue) and SAC (red) on the four environments we test in this work.

We use the -v3 version of the MuJoCo environments from OpenAI gym [22], which is the latest version available at the time of this research. We also use version 2.00 for MuJoCo itself. The state $\mathbf{s}$ and action $\mathbf{a}$ dimensions for the environments are:

- **Ant-v3**: state 111, action 8.
- **HalfCheetah-v3**: state 17, action 6.
- **Hopper-v3**: state 11, action 3.
- **Walker2d-v3**: state 17, action 5.

We build on top of the SpinningUp code [2] in this work to generate TD3 and SAC teachers (and students). For both algorithms, we use seed 40 to train HalfCheetah-v3 and Hopper-v3 teachers, and seed 50 to train Ant-v3 and Walker2d-v3 teachers (due to seed 40 resulting in poor TD3 performance in these environments), and save the resulting 8 data buffers to form the $\mathcal{D}^{(T)}$ datasets that students learn from in subsequent experiments.

Figure E.1 shows the TD3 and SAC teacher performance. The curves show the test time performance after each epoch by rolling out the (deterministic) policies for 10 episodes and averaging the resulting episodic rewards. Then we smooth the curves with a 20-window moving average. These results are comparable with prior results in the research literature [69, 57], though we caution that these works use -v1 of the MuJoCo environments, which may induce subtle differences.

E.2 Additional Experiment Details

For teachers and students, we use the default hyperparameters from SpiningUp for TD3 and SAC to ensure minimal changes from standard Deep RL implementations, as small adjustments can lead to significant performance differences [79]. The main exception to this is that we increase the default number of epochs from 100 to 250 to get 1M time steps.[1] The policy π and value Q networks are fully connected with two hidden layers, each of which have 256 nodes. The policy network outputs a vector in $\mathbb{R}^n$ where n is the action dimension, and the value function takes in the concatenated state and action $(\mathbf{s}, \mathbf{a})$ and output a single scalar in $\mathbb{R}$.

We note a subtle point regarding training and time steps: as mentioned in Section 7.2, we use the term "time step" to refer to both an online environment step and a gradient update, since in MuJoCo environments, the ratio between the two tends to be one [2]. However, this ignores a slight subtlety with the initial period of training, where it is common to perform several thousand environment interactions with a purely random policy to sufficiently populate a replay buffer before conducting gradient updates. The default in SpinningUp is to perform 1000 random steps,[2] which we use in this work. Thus, to make the students have the same number of gradient updates as a standard online RL agent, we start the student "time step" t at $t = 1000$. The students do one gradient update per time step up until $t = 1M$, assuming a standard run of 250 epochs. In this case, the first epoch will have 3000 gradient updates, and all the remaining 249 epochs have 4000 gradient updates.

[1] For the longer-horizon experiments from Section 7.5.2, we increase the epoch count to 2500.

[2] In SpinningUp, this is the `update_after` hyperparameter for TD3 and SAC.

E.2.1 Evaluation and Judging Performance with Standard Errors

We use a fixed set of random seeds (90 through 94) for students in all experiments, so any presented student results are aggregates over 5 independent learning runs initialized at those random seeds. Due to the limited number of trials, we do not report confidence intervals or bootstrapped estimates. Instead, we rely on the standard error of the mean, which is the sample standard deviation divided by $\sqrt{5}$. For tables in the main part of the chapter, we omit the standard errors to aid readability of student results, and instead report the standard errors throughout Section E.3.

For reporting results in tables, we compare among the effects of different data curriculum choices. The convention is to bold the best student M1 and M2 performance, which are separated by columns. *In addition*, we bold any other statistic which has overlapping standard errors as a fairer way of comparing different data curricula due to the highly stochastic nature of Deep RL [79]. More formally, consider a data curriculum for the student that results in the statistic for M1 (without loss of generality, M2) with the highest value, which is $w \pm x$ for standard error x. Then, for any other student data curriculum statistic, we have some performance $y \pm z$ with standard error z. By assumption, $w > y$. We do *not* bold face the result y if the following inequality holds:

$$w - x > y + z, \tag{E.1}$$

which suggests that even when subtracting the standard error from w and adding the standard error to y, the choice of data curriculum that led to the result of w still outperforms the one that produced y. We bold face the result y in tables if the inequality (E.1) does *not* hold, since that suggests the difference between the two statistics may not be significant.

E.3 Additional Experiment Results

We report additional results and statistics that we could not include earlier.

In tables here, we continue reporting M1 and M2 statistics (Section 7.4) and also report standard errors, *i.e.,* the sample standard deviation divided by $\sqrt{5}$. See Section E.2.1 for a further discussion of standard errors. In all figures in this chapter with student learning curves or other student statistics (*e.g.,* estimated Q-values), we shade in the standard error of the mean and smooth the curves over a moving average window of length 20 to improve legibility.

E.3.1 More Detailed Results from Section 7.5.1

We show additional learning curves for TD3 students learning entirely offline with different curricula, corresponding to experiments from Section 7.5.1. Figures E.2, 7.3, E.3, and E.4 show performance across 250 epochs for Ant-v3, HalfCheetah-v3, Hopper-v3, and Walker2d-v3, respectively. In Tables E.1 and E.2, we show the corresponding M1 and M2 statistics.

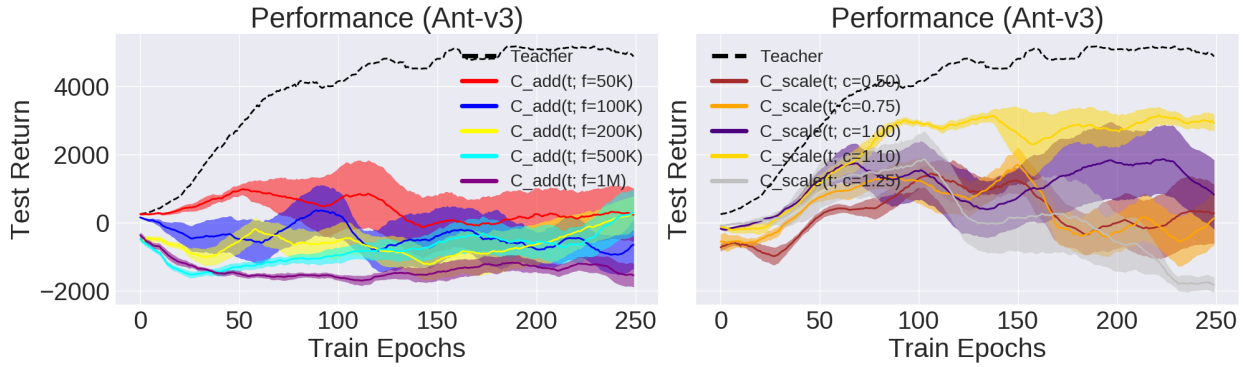

Figure E.2: Offline TD3 student performance on Ant-v3 based on additive curricula (left) and scale curricula (right); these are shown in the same way as Figure 7.3.

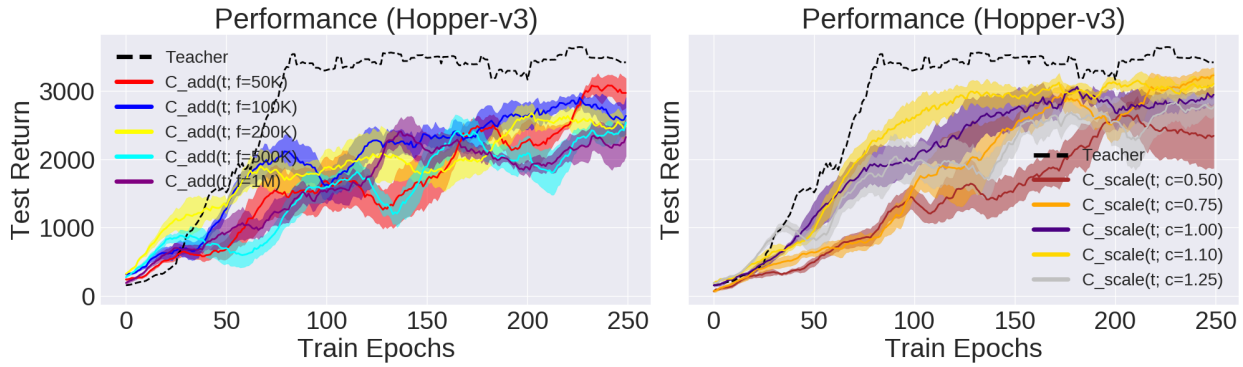

Figure E.3: Offline TD3 student performance on Hopper-v3 based on additive curricula (left) and scale curricula (right); these are shown in the same way as Figure 7.3.

We group results from Ant-v3 and HalfCheetah-v3, and then results from Hopper-v3 and Walker2d-v3. These are the same numbers in Table 7.1 except with standard errors now included.

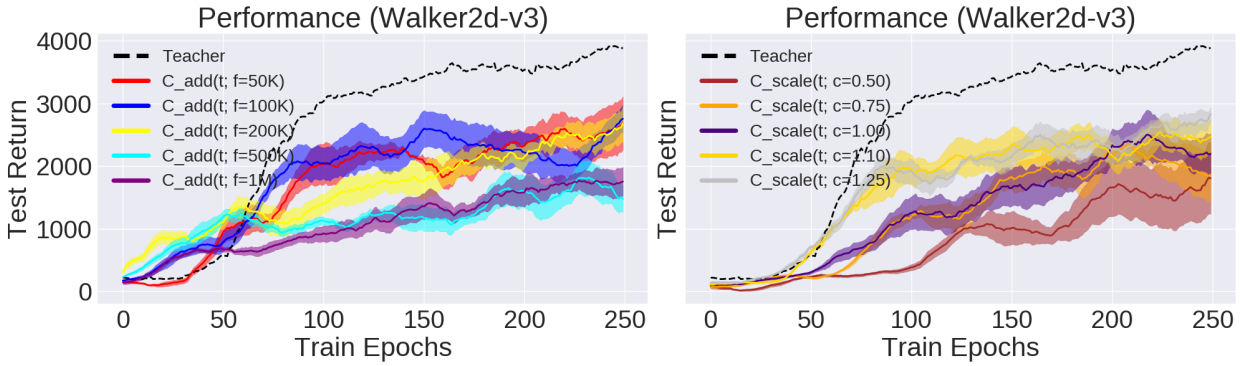

Figure E.4: Offline TD3 student performance on Walker2d-v3 based on additive curricula (left) and scale curricula (right); these are shown in the same way as Figure 7.3.

Table E.1: **Offline RL Results with DCUR**. M1 and M2 results for Ant-v3 and HalfCheetah-v3 students (and teacher in the bottom row), reported in a similar manner as in Table 7.1 but with standard error included.

	Ant-v3		HalfCheetah-v3	
Data Curriculum	M1	M2	M1	M2
$C_{\text{add}}(t; p = 1M, f = 50K)$	219.9 ± 765.0	382.0 ± 504.3	**8067.3 ± 123.6**	**7147.0 ± 45.7**
$C_{\text{add}}(t; p = 1M, f = 100K)$	-604.2 ± 724.9	-409.6 ± 420.5	7416.2 ± 540.2	**7052.6 ± 83.6**
$C_{\text{add}}(t; p = 1M, f = 200K)$	288.4 ± 454.4	-620.4 ± 385.4	**8028.4 ± 321.1**	6521.5 ± 138.7
$C_{\text{add}}(t; p = 1M, f = 500K)$	283.3 ± 667.1	-801.1 ± 254.5	7108.7 ± 449.6	5041.2 ± 307.2
$C_{\text{add}}(t; p = 1M, f = 1M)$	-1552.1 ± 354.0	-1390.5 ± 90.0	7447.3 ± 187.8	3846.8 ± 237.9
$C_{\text{add}}(t; p = 800K, f = 0)$	-889.7 ± 516.1	1497.9 ± 420.5	**7650.4 ± 806.4**	6942.7 ± 142.0
$C_{\text{scale}}(t; c = 0.50)$	285.0 ± 843.9	301.4 ± 318.9	7467.8 ± 52.3	6261.9 ± 83.9
$C_{\text{scale}}(t; c = 0.75)$	167.0 ± 791.9	412.2 ± 179.0	7392.6 ± 235.4	6689.2 ± 67.5
$C_{\text{scale}}(t; c = 1.00)$	825.6 ± 1022.4	1103.7 ± 628.9	**8305.3 ± 246.2**	6980.0 ± 111.6
$C_{\text{scale}}(t; c = 1.10)$	**2952.5 ± 217.5**	**2212.0 ± 194.4**	**8306.0 ± 255.1**	**7095.6 ± 79.4**
$C_{\text{scale}}(t; c = 1.25)$	-1851.1 ± 197.1	199.6 ± 515.2	7843.8 ± 150.2	**7175.4 ± 39.4**
TD3 Teacher	4876.2	3975.8	8573.6	7285.5

Table E.2: **Offline RL Results with DCUR**. M1 and M2 results for Hopper-v3 and Walker2d-v3 students (and teacher in the bottom row), reported in a similar manner as in Table 7.1 but with standard errors included.

	Hopper-v3		Walker2d-v3	
Data Curriculum	M1	M2	M1	M2
$C_{\text{add}}(t; p = 1M, f = 50K)$	2986.2 ± 191.3	1669.5 ± 141.3	**2715.6 ± 404.1**	**1712.1 ± 148.6**
$C_{\text{add}}(t; p = 1M, f = 100K)$	2627.4 ± 272.3	1980.1 ± 64.1	**2746.7 ± 231.3**	**1810.5 ± 191.5**
$C_{\text{add}}(t; p = 1M, f = 200K)$	2525.3 ± 173.0	1887.9 ± 99.1	**2691.8 ± 266.4**	**1651.9 ± 38.3**
$C_{\text{add}}(t; p = 1M, f = 500K)$	2498.0 ± 105.5	1546.2 ± 98.1	1515.7 ± 214.4	1183.5 ± 81.8
$C_{\text{add}}(t; p = 1M, f = 1M)$	2323.0 ± 323.0	1610.3 ± 77.3	1741.4 ± 226.8	1096.4 ± 76.9
$C_{\text{add}}(t; p = 800K, f = 0)$	2132.2 ± 255.2	1924.4 ± 132.5	792.0 ± 343.2	1521.8 ± 74.7
$C_{\text{scale}}(t; c = 0.50)$	2332.3 ± 467.5	1450.5 ± 86.2	1866.0 ± 589.1	799.9 ± 111.5
$C_{\text{scale}}(t; c = 0.75)$	**3284.7 ± 103.0**	1818.2 ± 101.9	1864.9 ± 410.8	1251.5 ± 71.4
$C_{\text{scale}}(t; c = 1.00)$	2984.3 ± 170.3	2092.6 ± 84.8	2178.5 ± 317.5	1305.7 ± 86.6
$C_{\text{scale}}(t; c = 1.10)$	**3185.2 ± 174.4**	**2317.1 ± 60.1**	2423.9 ± 278.7	**1698.1 ± 59.6**
$C_{\text{scale}}(t; c = 1.25)$	2755.5 ± 113.8	1891.9 ± 107.0	**2839.4 ± 126.6**	**1747.4 ± 85.8**
TD3 Teacher	3635.2	2791.9	3927.9	2579.8