Representation Reuse for Learning Robust Robot Manipulations

Mohit Sharma

Technical Report Number: CMU-RI-TR-24-03

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy.

February, 2024

Abstract

Real world robots need to continuously learn new manipulation tasks. These new manipulation tasks often share many sub-structures with previously learned tasks, e.g., sub-tasks, controllers, preconditions. In this thesis, we aim to utilize these shared sub-structures to efficiently learn new manipulation tasks. For this, we explore reusing skill representations. These skill representations are either provided manually as structured policy representations or learned in a data-driven manner.

The first part of this thesis focuses on policy representations. To learn *compositional* skill policies we propose object-centric task-axes controllers. Our task-axes controllers learn the skill structure and are composed into specialized policy representations for individual tasks. These representations utilize the compositional, object-centric and geometric structure underlying many manipulation tasks. As we show through extensive experiments, these representations are robust to environment variations and are learned from limited data. We also show how parameterized policy representations help learn new tasks efficiently in a lifelong learning manner. To achieve this, we propose skill effect models, which predict the effects of stereotypical skill executions. We utilize skill effect models together with the power of search-based planning to effectively plan for new tasks and learn new skills over time.

The second part of this thesis focuses on visual representations. These visual representations, learned either from simulation or offline web data are used for *efficient* learning of skill preconditions and policies respectively. Specifically, for skill preconditions we focus on compositional learning and show how complex manipulation tasks, with multiple objects, can be simplified by focusing on pairwise object relations. These relational representations are learned offline using large scale simulation data. In the latter part, we focus on skill policies that utilize large pretrained visual representations for robot manipulation. First, we propose RoboAdapters, which uses neural adapters as an alternative to frozen or fully-finetuned visual representations for robot manipulation. RoboAdapters bridge the performance gap between frozen representations and full fine-tuning while preserving the original capabilities of the pretrained model. Finally, we explore using large pretrained vision-language representations for real-time control of precise and dynamic manipulation tasks. We use multiple sensing modalities at different hierarchies to enable real-time control while maintaining the generalization and robustness of pretrained representations.

For my parents and all my teachers

Acknowledgements

The work presented in this thesis has only been possible by the support of the many amazing people I have met in the past few years. I acknowledge their support gratefully.

First, I would like to thank my advisor Oliver Kroemer. My foray into real-world robot learning began with Oliver taking me under his wing and I joined the Intelligent Autonomous Manipulation (IAM)[1] Lab. Before joining Oliver's lab I had almost never worked on a *real-robot* and by the end of my PhD I loved working on real-robot hardware. Oliver's mentorship was instrumental in my development as a robotics researcher. Over the past few years, I have learned not only the tools and technical aspects of robot manipulation and robotics in general, but what being a researcher entails—that is, patience, perseverance and kindness.

Oliver has always supported me in my research ideas. I am grateful for his patience and for giving me the opportunity to explore many different ideas with as much academic freedom as possible. Over the course of my PhD, I worked on many different projects and ideas and Oliver has provided me with both research guidance and encouragement for all of these projects. His guidance on asking the right research question, designing insightful and efficient experiments was invaluable. Oliver has an almost unparalleled knack of designing the right real-world experiment setup. Finally, I am grateful to him for having faith and confidence in me even when my ideas did not work.

Research is only one part of life and I would be remiss if I don't mention all the other ways in which Oliver has helped me in the past few years. He has always been there for any of our personal needs. From visa to health issues, he always prioritized personal well-being over everything else. He has always been very patient to my ramblings on the state of research to the lack of computing resources. Overall, Oliver's mentorship was a transformative experience. He not only helped me develop my technical skills but also instilled in me a sense of confidence and a love for research. I am incredibly grateful for his support and guidance.

I would also like to thank other members of my committee—Abhinav Gupta, David Held and Dieter Fox. I really appreciate Abhinav's guidance and his mentorship. He has always encouraged me to work on challenging and impactful problems. David's ability to question and deeply focus on the technical aspects of a research problem has been invaluable. His efforts on building a larger CMU robot manipulation community has helped me make new friends and be exposed to many rich and diverse ideas. I have immensely benefitted from the research papers coming from Dieter's group and lab. Dieter's ability to cut through the vast array of research and focus on the big picture has often guided me in my research.

I am also thankful to Kris Kitani for giving me the initial opportunity to pursue research. Working with Kris was a huge learning experience. The book reading sessions in Kris's lab meetings were always a great source of learning. Kris taught me *how to write* good research papers, structure my research problem statements and overall get a sense of joy from solving them.

Over the course of graduate school, I have been lucky to have many amazing mentors. I am grateful to Adith Swaminathan for providing me with the opportunity to work with him during an internship. He

[1]There is a word play here – I think therefore IAM.

taught me the value of focusing on important real-world problems and how to break them down into precise and manageable research questions. I am also extremely grateful to Jonathan (Jon) Scholz for giving me the opportunity to spend a great summer working with him and the many amazing people at DeepMind. Jon not only gave me the freedom to pursue what I thought was interesting but provided me with all the support needed to be successful. This support was also extended by many others at DeepMind. I am also extremely grateful to Yusuf Aytar for taking me under his wing and providing me with the opportunity to work closely with him. Despite his busy schedule, Yusuf was always able to take time out for research discussions and pair-programming sessions. His guidance and mentorship was invaluable and helped me grow as a researcher. I would also like to thank my other collaborators at DeepMind for all the brainstorming sessions, writing help and making my internship project a reality: Claudio Fantacci, Yi Yang, Mel Vecerik, Todor Davchev, Oleg Sushkov. I am also grateful to Chris Paxton and Vikash Kumar. Chris's passion towards robotics research and general enthusiasm towards real-world robots helping people is infectious. Vikash showed me how to carefully think about a new research statement in a very planned and systematic manner. He always finds time for any research or life discussion and I am extremely grateful for his guidance and advice.

I am also grateful to the wider CMU community, especially RI's administrative staff. Their presence and tiring effort is what makes our research possible. I would like to thank Suzanne Muth and Jean Harpley for always helping me with missed forms and academic requirements. I am grateful to Barbara Jean (B.J.) Fecich for making my initial CMU years easy.

Over the past few years I have been fortunate to be a part of two academic labs – Oliver's IAM-lab during my PhD and Kris Kitani's K-Lab during my Masters. Throughout this period I have been able to make amazing friends who have taught me much about research and life than I could have possibly imagined. I would like to thank my fellow IAMLab members – Alan Zhao, Ami Sawhney, Steven Lee, Tabitha Lee, Erin Zhang, Sarvesh Patil, Pat Callaghan, Mark Lee, Alex LaGrassa, Zilin Si, Xinyu Wang, Samuel Clarke, Austin Wang, Max Sieb. From KLab, I thank Nicholas Rhinehart, Xinshuo Weng, Xiaofang Wang, Shin Usami, Ye Yuan, Rawal Khirodkar, Aashi Manglik, Eshed Ohn-Bar, Harsh Agarwal, Yan Xu.

Throughout my time at CMU I have had the privilege to know and learn from many amazing people. I have listed the names of these people below, and I sincerely apologize in advance, if I have missed a name due to blatant oversight. At RI I would like to thank—Arpit Agarwal, Ashwin Khadke, Ratnesh Madaan, Thomas Weng, Wenxuan Zhou, Azarakhsh Keipour, Blake Buchanan, Shohin Mukherjee, Abhijat Biswas, Gaurav Mittal, Himangi Mittal, Ramkumar Natarajan, Jaskaran Grover, Sameer Bardapurkar, Kalyan Vasudev Alwala, Dominik Bauer, Swaminathan Gurumurthy, Adithya Murali, Pragna Mannam, Dinesh Reddy; fellow A-floor members: Arkadeep Chaudhury, Mrinal Verghese; fellow London-CMU* crew: Leqi Liu, Mohak Bhardwaj, Ben Eisner; friends within Machine Learning Department (MLD) CMU—Yusha Liu, Arundhati Banerjee, Ashwini Pokle.

I also want to thank fellow IIIT friends at CMU—Ishan Misra and Rohit Girdhar. I have known Ishan since undergraduate days and his dedication towards his work has always been inspiring. I also thank both Ishan and Rohit for their invaluable feedback and general advice during the time I applied to graduate school. Senthil Purushwalkam for providing me with useful feedback during multiple stages of my PhD. Gaurav Pathak and Siddharth Ancha for the many coffee sessions in the latter years of my PhD and for our shared disdain of the bright white rooms with no windows and the very short ceiling height in our

respective labs. I want to thank Gaurav Pathak for sharing his enthusiasm of running with me. Tanmay Shankar for the many discussions and his thought provoking questions. Nikolaos Gkanatsios (Nikos) for helping me navigate the large literature around visual learning for manipulation and helping me with his experiences on using some of this literature. Shivam Vats for our many discussions on what we need to do to improve our robots working in the real-world. Shivam often provides the sane voice in the room and helps me better contextualize our research musings.

Research is rarely a one person endevaor. One of the great joys during my PhD has been to bounce ideas off peers, learn from their experience, expertise and collaborate with them. While some collaborations did not end in a successful project, I learned from all of them in many different ways. I am grateful to many people for these collaborations. I would like to thank Kevin Zhang, with Kevin I collaborated on the first few projects of my PhD. Kevin is a jack-of-all-robots, his vast experience working with all the different kinds of robots made my initial robotics experience much easier. Kevin was also almost always extremely generous with his time, willing to help me with any of my hardware needs for any of of my projects. I would also like to thank Jacky Liang, with Jacky I worked on wrote two co-authored papers as part of this thesis. Both of these papers were possible due to Jacky's meticulous and incisive thoughts on what directions to pursue and how to formulate the right experiment and setting to make the project a reality. I would also like to thank Saumya Saxena with whom I worked on the last project of my thesis. Saumya's deep knowledge of controls and her love of physics had a greet influence on me. Her patience and insightful feedback in our many brain-storming sessions made the project a reality. I also thank Alex LaGrassa for being curious about many things, for helping me with any of my task and motion planning queries. I also deeply appreciate Alex's championing for robot teaching (whether it be undergrad mentoring or Girls-of-steel), robot ethics and our work's impact on society both within IAM-lab and at CMU.

I would also like to thank my collaborators from different labs. Within KLab I had the pleasure of collaborating with Arjun Sharma, Navyata Sanghvi and Shin Usami. With Arjun I had great fun working on many class projects and one research project. I deeply appreciate Arjun for introducing me to many different ideas and opportunities which greatly enriched my learning experience. Our late night coding sessions and our shared passion for soccer led to many impromptu virtual (FIFA) or real-world (Smith-Hall) soccer games. Overall, working with Arjun made research feel like a breeze. I would also like to thank Navyata Sanghvi and Shin Usami for involving me in their work of extending my masters project. Navyata and Shin were excellent collaborators, and working with them further enriched my own understanding of our work. During later stages of my PhD I also got the chance to collaborate with Homanga Bhardwaj and Jay Vakil. Homanga is a master of collabortion. During our collaboration he was always able to delineate the different part of the project that each of us should focus on, which made remote collaboration easy and fun. Jay took the burden of conducting most real-world experiments and the project would have been impossible without it.

I am also grateful to my many friends, both near and far, for supporting me throughout my PhD journey. My roommate, Aman Tyagi, for the first few years of my graduate school life. I have known Aman since undergraduate days and my initial days in Pittsburgh were made easy because of him. He took time to show me around and helped me settle into the academic rhythm. I am also grateful to Abhishek Bhatia, Ayush Dhingra, Abhilash Chowdhery, Aman Jain, for their companionship in Pittsburgh during these past few years. Abhilash's wedding in Pittsburgh was a really fun time which helped bring many of my

undergrad friends together. I am thankful to Jasmeet Singh, Ashish Massand, Oshin Paranjape for hosting me in San Francisco and Washington DC respectively. These visits helped me get away from the daily grind of a PhD life and allowed me to get a bigger perspective.

I would like to thank my partner, Tanya Marwah, for her constant support and guidance. She has been through the ups and downs of my PhD life and enriched it in ways too numerous to count. She taught me to stop and appreciate the small joys of life—be it cooking, coffee, or running. There are too many things I am grateful for but most importantly I am grateful to have her in my life.

Finally, none of this would have been possible without the constant support of my family. I would like to thank my brother, Rohit Sharma and his wife Aartika Rai. Their advice and feedback on many things outside research has helped me improve myself in tremendous ways. They often provide a clear reflection of my fractured reality, for which I will always be grateful. Last, but maybe most importantly, I would like to thank my parents who encouraged me to pursue my own dreams without the fear of failing. Their constant support, advice and wisdom has helped me gain perspective. It is their hard work and sacrifices that allowed me to get the best possible education and opportunities. While research stole many of my India visits, I hope to find more time now. It is impossible to fully express my gratitude for everything they have done for me. Their love and guidance are the most precious gifts I have received in my life.

Contents

List of Figures

List of Tables

Part I

Beginnings

We begin this thesis by first exploring the motivation for our problem statement as well as the particular research questions we tackle in Chapter 1. In Chapter 2-Chapter 3 we propose parameterized policy representations that utilize the compositional and object-centric nature of our physical world. These policy representations further utilize the shared sub-structure among manipulation tasks to learn robust policies for new tasks in a more efficient manner. In Chapter 4 we use parameterized skills to learn skill-effect-models and combine them with search-based planning for learning new tasks new tasks efficiently. Chapter 5- 7 focuses on reusing visual representations for manipulation tasks.

1 Introduction

The physical world we live in is *compositional* in nature. Much of the world consists of objects. Rigid objects can sometimes be broken down into finite parts, while deformable objects break down into infinite parts. Objects occupy space and can be combined with other objects to create new objects. Each object also has some state. Broadly, robot manipulation aims to *affect* this object state by interacting with the object.

The physical world is also highly *regular*. This regularity is expressed in many different ways from the 3D geometry of our surroundings, to the arrow of time and the law of gravity. Humans are remarkable in utilizing these regularities in our daily lives. Our perceptual and motor systems have evolved to make use of these regularities [215].

Moreover the physical world is not static and changes over time. Thus, the standard machine learning assumption of a static fixed dataset that encompasses the entire task distribution rarely holds in practice. By contrast, humans do not experience fixed scenarios and are known to be excellent incremental learners [228]. Our experiences (data distributions) change over time and we actively learn new skills *without forgetting* previously learned skills and simultaneously reuse them during our new experiences.

How can robots efficiently and continually accomplish new tasks in such a compositional and highly regular physical world? This thesis aims to answer a small part of the above question. For our first set of works we utilize the structure of manipulation tasks for efficient and robust learning. In the latter works, we explore shared skill models that avoid catastrophic forgetting while continually learning new tasks.

1.1 Role of Structure in Robot Manipulation

Robots in the real world will need to solve very diverse tasks such as cutting food, opening door, inserting screw. From a task-level perspective these tasks share little in common – each task involves a very different type of motion, the objects affected in each task are very different. Despite these differences, each of these tasks share many commonalities. For instance, each task's successful execution requires similar spatial reasoning (*common perceptual reasoning*), e.g., the food, door or screw need to be free of any obstacles to complete the task. Similarly, each of these tasks share many subtasks (*common subtask representations*), e.g., reaching or task-oriented grasping of the desired object. Finally, each task (or their associated subtasks) also share similar controllers (*common execution*), e.g., both cutting and inserting task requires appropriate force control at different stages of the task. Broadly, very different tasks are often modular with *shared sub-structure* across perception, planning and motor control aspects of these different tasks.

Figure 1.1: This thesis explores the questions on how can we effeciently reuse representations to learn new manipulation tasks more efficiently.

This shared sub-structure across tasks is important to learn new tasks both efficiently and robustly. By efficient learning we refer to the ability of robots to learn from few examples. Such sample efficient learning is crucial for real world systems where the cost of demonstrations (for imitation), or collecting new experience (for reinforcement learning), or programmatic implementation is prohibitively expensive. Being able to reuse or re-purpose the common sub-structure from previous tasks while learning new tasks is crucial for sample efficient learning.

In addition to sample efficient learning, task modularity also allows for robust task learning. Robust task learning is especially important for real world robot learning where we often have little training data and consequently very little overlap between the train and test distributions. Given this little overlap, and the inability of most function classes to extrapolate to out-of-distribution (OOD) scenarios, using monolithic functions to learn each task separately results in non-robust learning. On the other hand, with task modularity, the agent *only* needs to focus on the task aspect (sub-task or part of sub-task) that requires generalization. For instance, for the screw insertion task with randomized poses for the screw, the agent only needs to learn how to successfully align the screw and the tool (screw-driver). Once aligned similar force and rotation controllers should be sufficient to complete the task. Such modularity is often studied under the moniker of "compositional" learning.

1.2 Why Lifelong learning?

Robots in the real world (Figure 1.1) will operate in increasingly different scenarios over the course of their lifetime. These different scenarios result in changing data distributions over time. Traditional machine learning methods with static datasets and independent and identically distributed (i.i.d) assumptions often struggle with such changes in data distribution over time. While machines struggle, humans are well known to utilize continual learning [78, 248].

In addition to changing data distributions, robots also have to accomplish new tasks over time, while reusing the skills and data acquired from completing prior tasks. While appropriately reusing prior skills is important another significant challenge in new task acquisition is how to merge diverse skills with possibly very different parameterizations.

Finally, from a practical robot deployment perspective there exist other important aspects of continual learning. For instance, how to detect the change in data distribution for the robot to update it's model. While many works often assume streaming data and thus continuously update the model, real world scenarios may not even afford such continuous model updates. Another challenge in continual learning

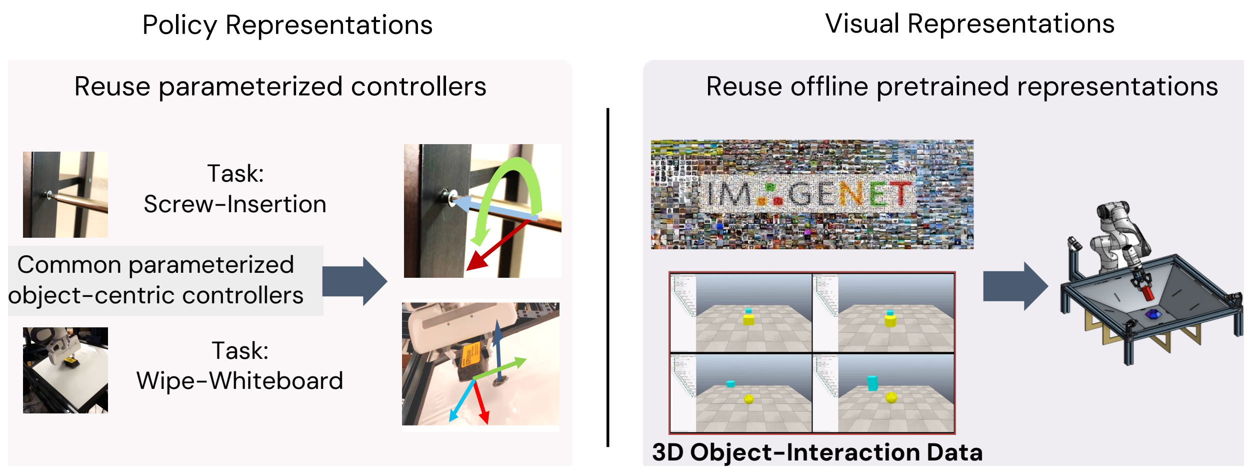

Policy Representations
Reuse parameterized controllers
Task: Screw-Insertion
Common parameterized object-centric controllers
Task: Wipe-Whiteboard
Visual Representations
Reuse offline pretrained representations
IMAGENET
3D Object-Interaction Data

Figure 1.2: This thesis shows how parameterized policy representations *(Left)* and data-driven visual representations *(Right)* can be used to efficiently learn robust manipulation policies.

is to ensure that the previous abilities of the model are not lost over time. While it is indeed possible to keep a subset of the data used to train the previous model, there exists no guarantees that training on this data together with new data will result in exactly similar capabilities as before. This possibly requires a data driven evaluation of the agent's capabilities.

1.3 Thesis Contributions

The contribution of this thesis (Chapter 2 to Chapter 7) lies along two different axes (Figure 1.2). Along the first axes, in Chapter 2 - Chapter 4 we focus on reusing parameterizing policy representations for learning manipulation tasks. In our next set of works (Chapter 5 - Chapter 7) we focus on visual representation learning.

Hierarchical Object-Centric Task-Axes Controllers: Chapter 2 focuses on compositional *skill learning*. Our main idea is based on the observation that many manipulation tasks often consists of multiple subtasks which often need to be performed *both in sequence and parallel*. For instance, consider the wiping task shown in Figure 1.1. To accomplish this task, the agent needs to focus on the surface-marks that need to be wiped while simultaneously maintaining contact with the table surface and applying a sufficient amount of force.

To achieve this we develop the idea of parameteric **task-axis controllers**. Our task-axes controllers contains both geometric and controller parameters. The geometric parameters model the underlying geometry of the scene via object keypoints, surface-normals, object-centers. We use multiple different controller implementations (e.g. position, force) and each controller contains its own set of parameters (e.g. force-targets for force-control).

We use this common set of parameteric task-axes controllers for all different manipulation tasks. For each manipulation task, we **automatically** instantiate a large set of controllers using our parameterized task-axes controllers. These controllers are composed both hierarchically and temporally to perform the given manipulation. Unlike prior works, we are the first ones to propose parameterized geometric controllers that can be instantiated automatically for each task. Further, unlike prior works we learn the appropriate task-axes controller to run at each step directly via interactions using reinforcement learning.

Skill Effect Models for Lifelong Robot Manipulation: Chapter 4 investigates how robots can solve new tasks with newly acquired skills while efficiently reusing prior learned skills. Prior works on planning with skills [22, 44, 104, 242, 247, 254] often make assumptions on the structure of skills and tasks, such as subgoal skills [247], shared skill implementations [134, 252], or task-specific plan skeletons [222, 254], which limit adaptation to new skills and tasks. By contrast, we propose doing task planning by jointly searching in the space of parameterized skills using high-level skill effect models learned in simulation. We use an iterative training procedure to efficiently generate relevant data to train such models. Our approach allows flexible skill parameterizations and task specifications to facilitate lifelong learning in general-purpose domains.

Relational Learning for Skill Preconditions: In Chapter 5 we aim to learn *skill preconditions* by using very little real-world data. Skill preconditions determine if a skill can be executed in any given environment [109, 111]. For robots to operate in dynamic and unstructured environments, these precondition models will need to generalize to new scenes containing variable number of objects with different shapes and sizes. Thus, we focus on learning precondition models for manipulation skills in unconstrained environments.

Our work is motivated by the intuition that many complex manipulation tasks, with multiple objects, can be simplified by focusing on less complex pairwise object relations. Prior works have shown the advantages of learning object relations for manipulation [3, 139, 190]. However, these works focus on simple discrete relations and use simple visual settings (i.e. with very few objects). By contrast, the key idea in our work is that we can learn high-dimensional object-relation representations by self-supervised learning in simulation and then transfer these learned relations for efficient precondition learning.

RoboAdapters – Lossless Adaptation of Pretrained Vision Models for Robot Manipulation: Chapter 6 focuses on robot perception for lifelong skill learning. Our work in Chapter 6 is motivated by the observation that large models pretrained on common visual learning tasks can provide useful representations for a wide range of specialized perception problems, as well as a variety of robotic manipulation tasks [61, 158, 171]. While prior work [158, 251] on robotic manipulation predominantly used frozen pretrained features, we demonstrate that in robotics this approach can fail to reach optimal performance, and that fine-tuning of the full model can lead to significantly better results. Unfortunately, fine-tuning disrupts the pretrained visual representation, and causes representational drift towards the fine-tuned task thus leading to a loss of the versatility of the original model.

We introduce **lossless adaptation** to address the feature disruption of classical fine-tuning. We demonstrate that appropriate placement of our parameter efficient adapters [5, 67] can significantly reduce the performance gap between frozen pretrained representations and full end-to-end fine-tuning without changes to the original representation and thus preserving original capabilities of the pretrained model.

Multi-Resolution Sensing for Real-Time Control with Vision-Language Models: Chapter 7 focuses on using pretrained vision-language models (VLMs) [4, 123, 179, 225] for robot manipulation. However, for many manipulation tasks large pretrained models can be very slow. Further, improving the inference of pretrained models without losing their existing capabilities is extremely challenging. Instead, in this work we leverage additional sensing modalities to complement the generalization capability of pretrained VLMs

We propose a multi-resolution architecture that uses *multi-spatial* as well as *multi-temporal* sensing modalities. Multi-spatial resolution sensing provides hierarchical information captured at different *spatial scales* (e.g. third-person camera captures global view while wrist-mounted camera and force-torque sensors capture very local information) and enables both coarse and precise motions. Multi-temporal resolution sensing uses each sensing modality at a different *temporal resolution*. Specifically, images from a static third-person camera change very slowly while force-torque information changes much more rapidly. By comparison, most prior works that use VLMs focus on quasi-static tasks only [219, 251], while works that perfom dynamic tasks do not focus on language guided generalization [154, 195, 216].

Overall, our framework learns generalizable language-conditioned multi-task policies that utilize sensing at different spatial and temporal resolutions using networks of varying capacities to effectively perform real time control of precise and reactive tasks. We leverage off-the-shelf pretrained vision-language models to operate on low-frequency global features along with small non-pretrained models to adapt to high frequency local feedback. We show that our multi-temporal resolution sensing enables the agent to exhibit high reactivity and real-time control.

1.4 List of Publications

- Chapter 2 Learning to Compose Hierarchical Object-Centric Task-Axes Controllers for Robotic Manipulation [213]
- Chapter 3 Generalizing Object-Centric Task-Axes Controllers using Keypoints [210] and Efficiently Learning Manipulations by Selecting Structured Skill Representations [209]
- Chapter 4 Search-Based Task Planning with Learned Skill Effect Models for Lifelong Robotic Manipulation [130]
- Chapter 5 Relational Learning of Skill Preconditions [212]
- Chapter 6 Lossless Adaptation of Pretrained Vision Models For Robotic Manipulation [208]
- Chapter 7 Multi-Resolution Sensing for Real-Time Control with Vision-Language Models [196]

1.5 Open-Source Contributions

During the course of this thesis we released some of our infrastructure code for ease of working with the Franka-Panda robots [263]. We released FrankaPy https://github.com/iamlab-cmu/frankapy which provides a simple to use python interface. FrankaPy is further accompanied with a C++ based interface which implements common robot controllers https://github.com/iamlab-cmu/franka-interface.

We have also released code for some of our other papers. We provide our implementation of hierarhical object-centric controllers Chapter-2 in https://github.com/iamlab-cmu/hierarchical-object-controllers. Further, we also open-source both our code and data for multi-resolution transformer policy (Chapter-7) in https://github.com/iamlab-cmu/mrest-multi-resolution-transformer.

Part II

Reusing Policy Representations

Robots in real world will need to solve a very wide range of diverse manipulation tasks. While each individual task can be different, its underlying components (e.g. subtasks, affected objects, controllers) are often shared with other tasks. In the first part of this thesis, we make use of this inherent compositionality of manipulation tasks. We propose a **shared** parameterized policy representation which can be composed to efficiently learn manipulation tasks [210, 213]. We further explore how parameterized policy representations can be used for efficient lifelong learning [130]. Specifically, we learn skill-effect-models (SEMs) which learn effects of stereotypical policy rollouts. We use SEMSs with search-based planning to compose existing skills and learn new skills for accomplishing varying tasks over time.

2 Object-Centric Task-Axes Controllers for Manipulation

This chapter is based on [Sharma, Liang, Zhao, LaGrassa, and Kroemer, 213].

Abstract: *Manipulation tasks can often be decomposed into multiple subtasks performed in parallel, e.g., sliding an object to a goal pose while maintaining contact with a table. Individual subtasks can be achieved by task-axis controllers defined relative to the objects being manipulated, and a set of object-centric controllers can be combined in an hierarchy. In prior works, such combinations are defined manually or learned from demonstrations. By contrast, we propose using reinforcement learning to dynamically compose hierarchical object-centric controllers for manipulation tasks. Experiments in both simulation and real world show how the proposed approach leads to improved sample efficiency, zero-shot generalization to novel test environments, and simulation-to-reality transfer without fine-tuning.*

2.1 Introduction

Manipulation tasks are inherently object-centric and often require a robot to perform multiple subtasks in parallel, such as pressing on a sponge while wiping across a surface, balancing a saucer while serving tea, or maintaining alignment of a screwdriver while unscrewing a screw. The individual subtasks need to be performed in parallel to accomplish the overall task. As the above examples illustrate, subtasks usually correspond to goals and constraints associated to objects in the robot's environment. Thus, manipulation skills are often defined as 3D motions, which are implemented as simple position or force controllers, of the end effector in object-centric coordinate frames.

One drawback of such an approach is that it results in monolithic controllers for each task, *i.e.* controllers which act specifically with respect to some fixed coordinate frame. In addition, for many tasks it is not always necessary to control all axes of a given object-centric coordinate frame. For instance, for the wiping task in Figure 2.1, the sponge needs to use the table surface normal to make contact with the surface, while it is free to move with respect to any other object (wall, corners, dirt) on the surface. Based on this insight, we adopt a modular approach by defining task-axis controllers for each potential subtask. Importantly, the controllers are associated with object-centric axes, such as the normal of a surface or the direction from the end-effector to an object.

We focus on learning an hierarchy of such object-centric task-axis controllers, or **object-axis controllers** (Figure 2.1). This hierarchy is especially important since many tasks require performing multiple subtasks in parallel. Previous works use pre-defined sets of task frames attached to objects or the robot, and they

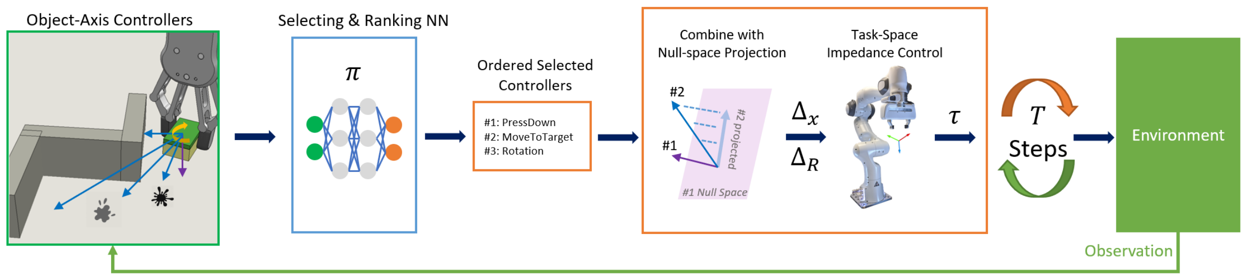


Figure 2.1: Controller Selection and Composition Pipeline. Given current observations and list of low-level controllers, an RL policy chooses an ordered list of controllers to use. These controllers are composed via nullspace projection, where the controls of lower-priority controllers are projected onto the nullspace of higher-priority ones. The combined control signals are used to actuate a robot via task-space impedance control. The controller combination runs for T time steps before the RL policy is queried again.

often learn a fixed task-frame hierarchy from human demonstrations. Instead, we use Reinforcement Learning (RL) to learn a policy that outputs an ordered list of controllers, which are then composed to be executed on the robot. To ensure different object-axis controllers do not interfere with each other, we compose controllers via nullspace projections [1], where the control signals of lower-priority controllers are projected onto the nullspace of higher priority ones.

In addition to modularity, our approach provides several other benefits. First, the object-axis controllers are not task specific, so they can be reused across multiple tasks. Second, composing controllers across multiple different objects makes the learned policies invariant to certain object properties *e.g.*, a controller that reaches toward an object is invariant to object size. Such invariances are useful for generalizing learned policies beyond the set of objects the policies are trained on. Finally, the use of a structured action space introduces meaningful inductive biases by ensuring robot actions are performed both in relation and with respect to objects in the scene. We successfully evaluated our approach on four different manipulation tasks, including two 2D tasks of fitting and pushing a block and two real robot tasks of screwing and door-opening. Experiments show that the proposed approach leads to improved sample efficiency, zero-shot generalization to novel environment configurations, and simulation-to-reality transfer without further fine-tuning. See videos and supplementary materials at `https://sites.google.com/view/compositional-object-control/`.

2.2 Related Works

Task Frames: Our use of task-axes is related to the notion of 6D task frames [11, 12, 142]. One of the first works to formalize task frames is [142]. There, the authors referred to different task-axes as compliant or non-compliant based on the type of desired motion along each axis. The authors of [181] proposed hybrid force-position control, which selects different axes of the constraint frame for either position or force control. Simultaneously, the authors of [11, 12] proposed task frames to define robotic manipulation primitives; they noted that the geometric level of task frames can serve as a good middle ground between symbolic actions and the motor control input. Since then, task frames in the form of task spaces have been used extensively in robotics [203]. Prior works treat task-frames as fixed coordinate frames which are either attached to objects of interest or generated from constraints in the environment. By contrast, our approach is more modular and dynamic, as it enables an RL policy to combine task-axes across different objects and dynamically synthesize task-frames.

Task Frame Selection: Although the use of task frames and spaces is widespread in robotics [17, 100, 103, 138, 151, 155, 243], only a few works have explored using learning to select which task frames are appropriate for the given task [31, 103, 155, 177, 243]. However, most of these works use imitation learning *i.e.*, they learn task frame selection from human demonstrations [103, 155, 177, 243]. The criterion for task-frame selection is typically manually defined using properties such as inter-trial variance or convergence behavior of demonstrations. In our work, we set task-axes selection as the action space for an RL agent, so we do not require demonstrations. Moreover, the RL agent chooses a hierarchy of task-axis controllers, which are composed together for execution.

Hierarchical Controllers: Combining multiple task-axis controllers is related to works in hierarchical control. Hierarchical control is often used in robots with redundant degrees of freedom or bi-manual robot setups where multiple tasks or objectives can be executed in parallel [41, 92, 96, 159]. To combine different controllers, these works project the control signals of lower-priority controllers onto the nullspace of higher-priority controllers. However, most of these works assume a fixed priority order for the tasks/objectives being considered, while some recent works [92] learn the priorities from human demonstrations. Similar to these works, our approach also uses nullspace projections to combine multiple task-axis controllers together. However, instead of using a fixed priority order, our method learns to prioritize controllers by directly interacting with the environment.

Reinforcement Learning: Finally, our approach is related to works on structured action spaces for reinforcement learning (RL) for contact-rich manipulation tasks. Recent works have studied how the choice of action spaces affect robot learning performance [16, 18, 141]. However, these methods focus only on the final controller output, *i.e.*, comparing fixed with variable impedance control [18, 141] or with hybrid-force position control [16] in joint and task-spaces. Our work provides additional structure to the action space via composing hierarchical object-centric controllers.

Hierarchical RL: Composing task-axes controllers for performing tasks is also related with hierarchical RL (HRL) [30, 105, 238]. HRL uses the notion of options, which are temporally-extended actions, and learns to combine them to accomplish a given task. There has been a large body of work which aims to extract the underlying options [9, 224, 233], using techniques such as bottleneck states [233], policy sketches [217], or expert demonstrations [35, 108, 214]. Similarly, there have also been works that use predefined option policies and compose them to learn a “meta-policy" [131]. However, these option policies are defined specifically with respect to the underlying task, and hence it is not clear how reusable these policies are. By contrast, our proposed task-axes controllers are reusable across multiple different manipulation tasks. This is desirable for efficient learning of new manipulation tasks[153]. Additionally, task-axes controllers are different than options since they can be composed both hierarchically and temporally.

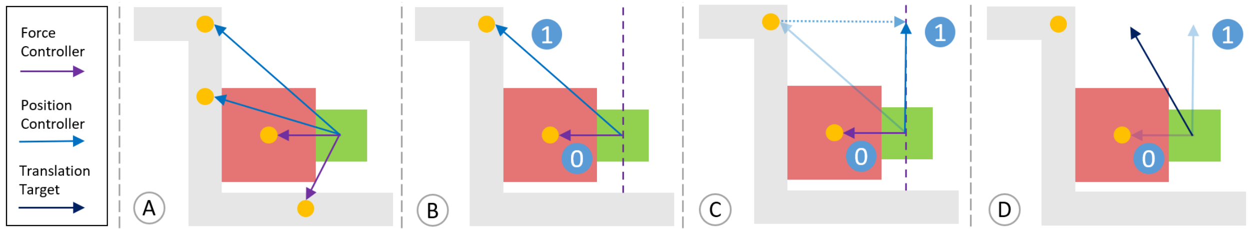


Figure 2.2: Force-Position Controller Composition. Here, the agent controls the green block to push the red block up along the vertical gray wall. A) The agent is given 4 controllers to choose from, each corresponding to points of interests in the scene. B) The agent chooses 2 controllers, with the force controller into the red block at the higher priority (0), and position controller toward the wall corner at the lower priority (1). C) The error of the lower-priority position controller is projected onto the null space of the higher-priority force controller (purple dashed line). D) The projected errors are combined to form the desired position target.

2.3 Learning Hierarchical Compositions of Object-Centric Controllers

We propose training an RL policy to perform manipulation tasks by using a structured action space consisting of hierarchical compositions of object-centric controllers. Each object in the scene is associated with a fixed set of task-axes, positioned either at object centers or other object key points. For each axis, we define a set of controllers that perform force, position, and rotation controls. This gives a set of predefined object-centric task-axis controllers, or object-axis controllers, which define our structured action space. With this action space, instead of directly commanding the end-effector, the RL policy selects multiple object-axis controllers in a prioritized order, which are composed together using null-space projections. Figure 2.1 shows an overview of the overall proposed approach.

In the next subsections, we first define the different types of object-centric low-level controllers we use, including how their object-centric axes are defined. We then discuss how to combine different object-axis controllers together using null-space projections. Finally, we discuss different RL approaches for learning the high-level policy that selects multiple controllers.

2.3.1 Controller Types

In this work, we use three different types of controllers: position, force, and rotation. These controllers are object-centric, *i.e.* their control targets and axes correspond to objects in the scene. For example, position controllers could be attractors that lead the end-effector (EE) close to an object of interest, force controllers could be applying forces perpendicular to object surfaces, and rotation controllers could be aligning an axis of the EE with an axis of the object. Currently, these controllers are manually specified (see details in Section 2.4), but they could also be autonomously inferred from visual observations of objects in the environment. Figure 2.2 illustrates force and position controllers and their composition, and Figure 2.3 shows the rotation controllers.

Let $x_c \in \mathbb{R}^3$, $R_c \in SO(3)$, and $f_c \in \mathbb{R}^3$ respectively denote the current end-effector position, orientation, and forces expressed in the robot's base frame.

Position and Force Controllers: The position controller consists of a target position x_d and an axis u along which the controller will move the robot's end-effector toward the target. u can be a fixed direction, like the normal direction of a surface, or it can be adapted with respect to x_c: $u = \frac{x_d - x_c}{\|x_d - x_c\|_2}$. Let $\mathcal{P}(u) = uu^\top$ be the projection matrix for the given axis. Then, the translation error a position controller produces is defined as $\delta_x(x_d, u, x_c) = \mathcal{P}(u)(x_d - x_c)$. The force controller is similar to the position controller, *i.e.* given a force target f_d and an axes-direction u, the force error the controller produces is $\delta_f(f_d, u, f_c) = \mathcal{P}(u)(f_d - f_c)$.

Rotation Controller: The rotation controller attempts to align one axis $R_c u$ of R_c with a target axis r_d, where u is a unit vector that performs axis-selection. For example,to align the X-axis of the end-effector frame to align with r_d, then $u = [1, 0, 0]^\top$. The rotation controller produces a delta rotation target in the end-effector frame, which we compute via the angle-axis representation: $\delta_R(r_d, u, R_c) = \cos^{-1}((R_c u)^\top r_d)((R_c u) \times r_d)$

Null Controllers: The high-level policy also has the option to choose a null controller, which would give 0 errors for both δ_x and δ_R. While other controllers can be chosen at most 1 time, the null controllers can be chosen multiple times, giving the high-level policy more flexibility.

2.3.2 Controller Composition

Force-Position Composition: The RL policy selects at most 3 force and position controllers to compose. Only 3 of force and position controllers can execute concurrently, because there are only 3 position dimensions. The RL policy outputs a priority order for these controllers. Let the indices $[0, 1, 2]$ denote the 3 controllers in decreasing priority, so 0 is the highest, and 2 the lowest. The final position target is computed by projecting the lower-priority targets onto the nullspaces of the higher-priority controllers, then summing them. Let $\mathcal{N}(U) = I - U^\dagger U$ be a nullspace projection matrix with respect to rows of U, where † denotes the pseudoinverse. Let K_x be the position controller gain and K_f the force gain:

$$\Delta_x^0 = K_x \delta_x(x_d^0, u^0, x_c) \tag{2.1}$$
$$\Delta_x^1 = K_x \mathcal{N}([u^0]) \delta_x(x_d^1, u^1, x_c) \tag{2.2}$$
$$\Delta_x^2 = K_x \mathcal{N}([u^0, u^1]) \delta_x(x_d^2, u^2, x_c) \tag{2.3}$$
$$\Delta_x = \sum_{i=0}^{2} \Delta_x^i \tag{2.4}$$

where $[\ldots]$ represents a concatenation operator, *i.e.* concatenation of vectors into a matrix, *e.g.*, $[u^0, u^1] \in \mathbb{R}^{2\times 3}$. Although the above expressions are written with all 3 controllers as position controllers, in our implementation we combine multiple position and force controllers together. If force controllers are used, for the corresponding controller, swap δ_x with δ_f, x_d with f_d, x_c with f_c, and K_x with K_f. Figure 2.2 illustrates the force-position controller composition.

Rotation Composition: The RL policy selects at most two rotation controllers to compose. This is because when the highest priority controller fixes one axis of a rotation frame, there is only one degree of freedom left, which is a rotation in the 2D nullspace of the fixed axis. Similar to force-position controller

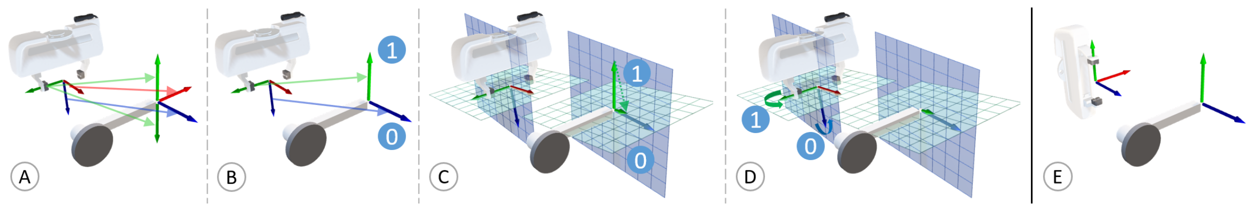


Figure 2.3: Rotation Controller Composition. Here, the agent rotates the Franka robot's gripper from the initial pose (A) to the final pose (E), so the gripper aligns with a door handle. A) The agent is given 4 rotation controllers to choose from, aligning various axes of the gripper with different target axes of the handle. B) Two controllers are chosen with the higher-priority labeled as (0) and the lower-priority as (1). C) Both the current and target axes of the lower-priority controller (green arrows) are projected down to the null-space (green planes) of the current axis of the higher-priority controller (gripper's blue axis). D) The desired rotation target is formed by combining the higher-priority rotation in the blue plane with the projected lower-priority rotation in the green plane. Note that the lower-priority rotation does not interfere with the higher-priority rotation.

compositions, we project the errors of lower-priority controllers onto the nullspace of higher-priority controllers:

$$\Delta_R^0 = K_R \delta_R(r_d^0, u^0, R_c) \tag{2.5}$$

$$\Delta_R^1 = K_R \delta_R(\mathcal{N}([R_c u^0]) r_d^1, u^1, \mathcal{N}([R_c u^0]) R_c) \tag{2.6}$$

$$\Delta_R = \Delta_R^1 \circ \Delta_R^0 \tag{2.7}$$

where $\circ$ denotes composing rotations, and K_R denotes a rotation error gain. This procedure ensures the higher-priority rotation controller always reaches its goal, and the trajectory of that axis is not affected by the lower-priority controller (see Figure 2.3 for an illustration).

Controlling the Robot: We use task-space impedance control to convert translation and rotation targets to configuration-space targets via Jacobian transpose, and we actuate the robot via joint torques. We first concatenate the translation target Δ_x with the axis-angle representation of Δ_R to form the final 6D delta end-effector target Δ. Then, the robot joint-torque commands are computed as $\tau = J^\top(K_S\Delta + K_D\dot{\Delta})$, where K_S and K_D are diagonal stiffness and damping matrices, and J is the analytic Jacobian. Terms for compensating gravity and Coriolis forces are omitted for brevity. In practice, we cap the magnitude of Δ to limit maximum control effort, and we add an integral term to the force controllers for better convergence. Once a set of controllers are selected, their combination runs for T timesteps before the RL policy is queried again for a new set of controllers.

2.3.3 RL with Object-Axis Controllers

We use RL to learn a policy that composes object-axis controllers to perform the underlying task. The policy outputs an ordered list of controllers, which are composed together to output the final control signal to move the robot. The combination of controllers is run for a fixed T timesteps, before the RL policy is queried again. Note that the controllers do not have to converge before the RL policy switches to the next combination. We next discuss multiple ways in which the RL policy can output the ordered list of controllers.

Discrete Combinatorial Actions: Let N be the total number of available controllers, and N_c be the number of controllers that can be executed simultaneously. One simple way to output an ordered list of N_c controllers is to use a discrete action space, where the policy selects an action from all available controller permutations. Such an action space grows combinatorially ($\mathcal{O}(N^{N_c})$), and is not scalable for environments with a large number of controllers.

Continuous Priority Scores: A continuous space alternative is to allow the policy to output a priority score in $[0, 1]$ for all controllers. These priority scores are then used to order the controllers, where the N_c controllers with highest priorities are executed at each step. Although the dimension of this action space grows linearly with the number of controllers, it can often lead to sub-optimal performance since the agent now needs to explore a much larger action space than before.

Expanded-MDP: To avoid the sub-optimal performance of the above methods, we propose an expanded-MDP formulation that still uses a discrete action space while avoiding combinatorial expansion. Here, we expand each environment-execution step of the MDP into N_c intermediate controller-selection steps, with the original environment-execution step occurring after the N_c'th intermediate step. At each intermediate step, the policy selects one controller from the N choices. Once N_c controllers are selected, the robot takes an actual environment step. The reward function is modified such that 0 rewards are given for the controller-selection steps before the N_c'th step. Similar MDP transformations have been suggested previously to solve continuous action MDPs using discrete action space RL algorithms [150, 175].

To use the Expanded-MDP formulation, at each controller-selection step the policy needs to know its previous controller selections. One approach is representing each controller with 1-hot encoding and appending the 1-hot encodings of previously selected controllers to the observations. This expands the observation space by $N \times (N_c - 1)$ dimensions, and we refer to this representation as **multi-1-hot**. However, in many cases it might not be necessary to know the order of the previous controllers being selected, *i.e.*, it is sufficient to know which controllers have been selected previously but not their order. So, for the second representation, we merge the one-hot encodings of multiple previous controllers into one binary vector. This only increases the observation space by N dimensions, and can lead to faster learning. We refer to this representation as **single-1-hot**

2.4 Experiment Tasks and Setup

With our experiments we aim to evaluate 1) How useful are the proposed object-axis controllers for task learning, 2) How important is controller composition for task learning, and 3) How well does our proposed approach generalize to the different test configurations.

Figure 2.4 visualizes the tasks used to evaluate our approach. There are two 2D tasks, Block Fit and Block Push, and two real robot tasks, screwing hex-screws and opening doors with the 7 DoF Franka Emika Panda arm. We compare both learning performance of the proposed approach against baselines, as well as their ability to generalize to novel environment configurations. To study generalization, we train

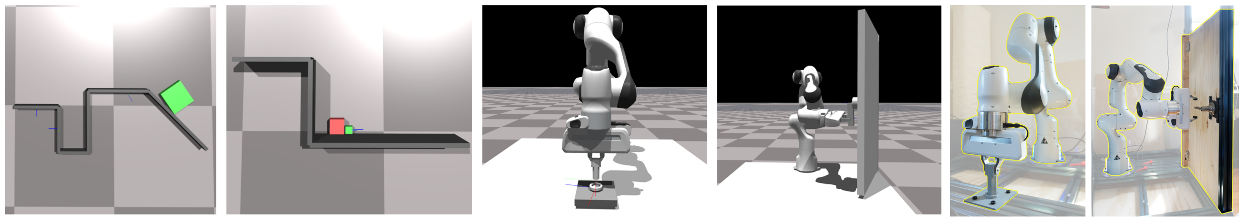


Figure 2.4: Experiment Tasks. From left to right: Block Fit, Block Push, Franka Hex-Screw, Franka Door-Opening tasks implemented in simulation, and Franka tasks in the real world.

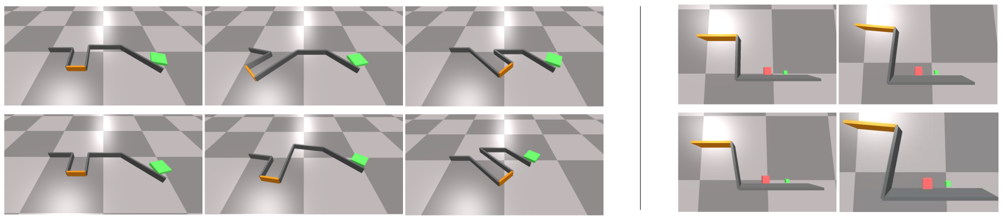


Figure 2.5: Example environment configurations for Block Push (left) and Block Fit (right) environments. Top row shows *some* examples of train configurations, and the bottom row shows *some* examples of test configurations. The orange wall shows the goal wall to reach.

policies on a small set of training environment configurations and test them on a novel test set. Training over multiple environments is important to avoid overfitting. Details of each task, including controller specifications, task variations, observation and action spaces, and the reward functions can be found in the Appendix.

Block Fit: In this task, a 2D block robot needs to navigate to a 2D goal pose in the scene. There are multiple walls or obstacles in the scene, so the robot cannot directly proceed towards the goal. Figure 2.5 (Left) shows *some* of the different train and test configurations. The low-level controllers are wall-centric. Different environment configurations have different wall lengths and angles between walls. The training set has 8 different environment configurations, while the test set has 9.

Block Push: In this task, a 2D block robot needs to push another block along a vertical wall over a ledge to a desired goal pose. Figure 2.5 (right) visualizes *some* train and test configurations. Controllers and environment wall configurations are similar to those of Block Fit. The environment samples the initial pose of the block robot and the target block. The training set has 11 different environment configurations, and the test set has 8.

Franka Hex-Screw: In this task, a 7-DoF Franka Panda arm is used to insert a hex-key into a screw, and turn the screw to a desired angle while applying a downward force and maintaining vertical orientation. The screw will not turn unless a sufficient pre-defined $(20N)$ downward force is applied. Different environment configurations have different wrench and screw sizes. The training set uses size scale multipliers of $(0.9, 1.0, 1.3)$, and the test set uses $(0.7, 0.8, 1.1, 1.2, 1.4, 1.5)$.

Franka Door-Opening: In this task, the Franka robot needs to open a door by first turning its door handle and then pulling the door beyond an opening threshold. To avoid trivial policy solutions, the door will not open unless the handle is first turned to a desired angle. The environment samples the

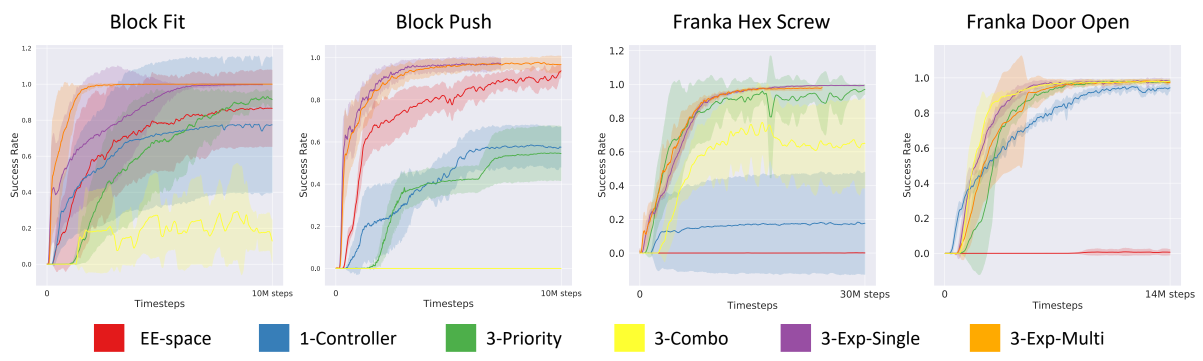


Figure 2.6: Success rates for all tasks on training environment configurations.

initial relative pose between the EE and the door, and different configurations have different locations of the door handle on the door. The training and test set contain 4 and 3 configurations.

Compared Approaches: We set $N_c = 3$ across all experiments, which we found to be sufficient. To evaluate the utility of our proposed object-axis controllers we compare against an RL agent that controls the robot directly via end-effector delta-poses. We call this approach **EE-Space**. We also evaluate the need for executing multiple controllers in parallel by comparing against a baseline which only chooses 1 controller at each timestep. We call this **1-Ctrlr**. To show the efficacy of our proposed Expanded-MDP formulation we compare against both: discrete combinatorial (**3-Combo**) and continuous priority scores (**3-Priority**) action spaces. Both these approaches naively combine all possible controller combinations and we show how this can lead to sub-optimal performance.

RL Training: We use Proximal Policy Optimization (PPO) [201] implemented in stable-baselines [65] across all tasks and action space variants. Given the high variance in policy-gradient RL algorithms, we run all methods with 8 different seeds (sampled uniformly between 1 and 100). All tasks are simulated with an NVIDIA Isaac Gym [1], a GPU-accelerated robotics simulator [129].

Metrics: We report the success rates of the learned policies separately for train and test environment configurations. Performance on the train set indicates whether or not the approach can robustly solve a task, and performance on the test set evaluates generalization abilities. Test set is split into two subsets, one with small deviations from the train configurations, and another with larger deviations. We report additional results including more fine-grained analysis for each task in the Appendix.

2.5 Experiment Results and Discussion

Block Tasks: Figure 2.6 (left) plots the success ratios averaged over all train environment configurations for Block Fit and Block Push. The Expanded-MDP methods are able to successfully learn both tasks. While EE-Space also makes progress on both tasks, it has a lower success rate, and this is due to its inability to robustly solve a few challenging configurations (see Appendix). Both 1-Ctrlr and 3-Priority perform well on Block Fit but poorly on Block Push. We attribute this difference to how there is a greater need

[1] https://developer.nvidia.com/isaac-gym

Task	Variation	EE-Space	1-Ctrlr	3-Priority	3-Combo	3-Exp-Single	3-Exp-Multi
Block Fit	Train	0.87 (0.213)	0.778 (0.38)	0.936 (0.032)	0.294 (0.18)	0.998 (0.002)	**1.00 (0.0)**
	Test-Small	0.87 (0.10)	0.916 (0.14)	**0.99 (0.001)**	0.184 (0.12)	**0.99 (0.001)**	**0.99 (0.01)**
	Test-Large	0.371 (0.246)	0.396 (0.423)	0.877 (0.141)	0.165 (0.23)	**0.974 (0.048)**	0.953 (0.087)
Block Push	Train	0.966 (0.046)	0.594 (0.087)	0.548 (0.129)	0.0 (0.0)	0.974 (0.025)	**0.978 (0.022)**
	Test-Small	0.912 (0.045)	0.577 (0.193)	0.396 (0.041)	0.0 (0.0)	0.945 (0.045)	**0.960 (0.030)**
	Test-Large	0.518 (0.185)	0.152 (0.137)	0.376 (0.032)	0.0 (0.0)	0.751 (0.103)	**0.788 (0.132)**

Table 2.1: Mean (SD) success rates for Block Fit and Block Push tasks on different environment configurations.

Task	Variation	EE-Space	1-Ctrlr	3-Priority	3-Combo	3-Exp-Single	3-Exp-Multi
Hex-Screw	Train	0.002 (0.002)	0.183 (0.303)	0.960 (0.048)	0.774 0.194)	**0.984 (0.01)**	0.980 (0.016)
	Test-Small	0.00 (0.00)	0.13 (0.072)	0.62 (0.045)	0.429 (0.430)	0.963 (0.01)	**0.966 (0.015)**
	Test-Large	0.00 (0.00)	0.026 (0.025)	0.633 (0.081)	0.34 (0.057)	**0.936 (0.028)**	**0.936 (0.035)**
	Real-World	n/a	0.0	0.5	0.0	**0.9**	0.6
Door-Open	Train	0.002 (0.006)	0.947 (0.021)	0.982 (0.007)	0.984 (0.013)	**0.987 (0.009)**	0.984 (0.015)
	Test-Small	0.066 (0.063)	0.922 (0.043)	0.965 (0.046)	0.975 (0.011)	**0.997 (0.006)**	0.992 (0.015)
	Test-Large	0.000 (0.001)	0.936 (0.032)	0.983 (0.006)	0.985 (0.007)	**0.996 (0.005)**	0.994 (0.013)
	Real-World	n/a	0.0	**1.0**	0.9	**1.0**	**1.0**

Table 2.2: Success rates for Franka Hex-Screw and Open-Door tasks on train and test environment configurations across 8 seeds. Parentheses denote standard deviation. Real-world results are evaluated over 10 trials each. We did not run EE-Space policies in the real world as they were unable to learn the tasks in simulation.

to use multiple controllers in the right order for Block Push. For instance, the policy needs to choose a force/position controller that pushes into the wall and then another controller to move up. In addition, robustly pushing the block around the edge of the vertical wall also requires multiple controllers. Although it is feasible to achieve this by quickly switching between controllers, such a strategy is not robust. 1-Ctrlr is unable to use multiple controllers at the same time, and using the high-dimensional priority score action space is challenging.

Table 2.1 shows success rates for both tasks on two sets of test configurations. Both EE-space and Expanded-MDP methods perform well when test configurations have small deviations from train configurations, with EE-Space performing slightly worse. However, for large deviations, EE-space performs poorly, achieving success ratios of 0.371 for Block Fit and 0.518 for Block Push. By contrast, Expanded-MDP methods perform much better, achieving 0.974 for Block Fit and 0.788 for Block Push, and 3-Priority also outperforms EE-Space for the Block Fit task. In addition, 1-Ctrlr sees greater performance degradation going from small to large deviations in test configurations. Together, these results indicate that using a structured action space of multiple object-centric controllers leads to better generalization than using one controller or directly learning in the EE-space.

Franka Tasks: Figure 2.6 (right) shows training results for both Franka Hex-Screw and Door-Open tasks. The Expanded-MDP methods perform well on both the tasks, while EE-Space does not make progress on either task. For Hex-Screw, the EE-Space policy is able to reach the screw, but is unable to learn to simultaneously rotate the screw and apply sufficient downward force. For Door-Open, the EE-Space policy reaches the door handle, but fails to grasp and completely rotate the door handle in a robust manner to open the door. One reason for these EE-Space failures is that exploration in both tasks is difficult in the end-effector space. To aid EE-Space exploration, we evaluated the approach from [173], which gives the agent additional exploration rewards. While doing so leads the agent to cover a larger region in the state space, the explored states do not always correspond with meaningful behaviors for task completion, so we did not observe any gains using this method.

Unlike with the Block 2D tasks, 3-Priority is able to learn both the Franka tasks. This is because the Franka tasks have fewer possible controllers, which resulted in lower dimensional priority-score action spaces. The reduced action-space dimensions of Franka tasks allowed us to evaluate 3-Combo, which is also able to learn both tasks, although it achieves worse performance on Hex-Screw. Similarly, 1-Ctrlr is able make progress on Door-Open but not Hex-Screw, which suggests that Hex-Screw requires more precise coordination of multiple controllers than Door-Open. Table 2.2 (rows 2, 3, 5 and 6) shows the success rates for both tasks on test configurations with small and large deviations. All methods that use hierarchical combination of multiple object-axis controllers generalize well to both small and large test deviations. Methods that performed poorly during training, EE-Space for both tasks and 1-Ctrlr Hex-Screw, do not generalize well.

To evaluate Franka tasks in the real-world, we performed 10 trials of each method on the real robot, each trial with a different sampled initial state. For the Hex-Screw task, we further tested on 3 different screw and key sizes. All methods that used the proposed composition of hierarchical controllers were able to robustly perform Door-Open in the real world, while only 3-Exp-Single was able to do so for Hex-Screw. Hex-Screw is more challenging than Door-Open, because it requires more precise movements for alignment and insertion. As a result, sim-to-real gap in the robot dynamics and controller responses leads to greater performance degradation for Hex-Screw than for Door-Open.

2.6 Conclusion and Future Work

In this work, we propose using RL to learn how to compose hierarchical object-centric controllers for manipulation tasks. Our approach has several advantages. First, the object-centric controllers can be reused across multiple tasks. Second, controller compositions are invariant to certain object properties. Finally, the use of a structured action space introduces meaningful inductive biases for manipulation. Our experiments show that the proposed approach leads to more guided exploration and consequently improved sample efficiency, and it enables zero-shot generalization to test environments and simulation-to-reality transfer without fine-tuning. In future work, we will tackle the main limitations of the current approach – the set of controllers is fixed and manually-defined.

3 Efficiently Learning Generalizable Manipulations using Task-Axes Controllers

This chapter is based on [Sharma and Kroemer, 210] and [Sharma and Kroemer, 209].

3.1 Introduction

Manipulation tasks in real world involve objects of varying and often unknown shapes, sizes and geometry. Learning manipulation skills to successfully perform tasks across a wide range of objects, without access to their underlying geometric models, is a challenging problem. Recent work has shown how simple keypoint representations can obviate the need of known geometric models [49, 140]. These keypoint representations which are learned purely from visual data are easy to acquire, and importantly provide accurate and robust intra-category generalization capabilities. However, the keypoint representations as used in [140] are only used for 1-step actions, which are found as a solution of an optimization problem. Additionally, although keypoint representations have been used for Imitation Learning (IL) [48], they have only been used for state representations, which are used as inputs to monolithic neural networks and thus do not encode any semantic information (*e.g.* handle of a cup) about the object and the task being performed. In this chapter, we focus on learning such generalizable manipulation skills by possibly utilizing the semantic structure of the task.

A key challenge for learning generalizable manipulation skills is to choose an appropriate skill representation. For instance, we can use monolithic neural-network controllers [110, 122] to generate trajectories or low-level actions for many different manipulation tasks. On the other hand, we can also use more specialized skill representations, *e.g.*, parameterized force-controllers [181], object-centric attractors [52, 213], or Dynamic Movement Primitive (DMP) [71]. While general skill representations may enjoy widespread applicability their over-parameterization can make them unnecessarily difficult to learn. More specialized skill representations often work well in only limited settings, but they can be faster to learn and adapt given their fewer parameters. The robot should therefore ideally use the most specific skill representation that is still suitable for the given task.

Ov3erall

However, [213] does not use any visual data in their formulation. Instead, they use heuristics to define the set of possible controller parameterizations for each task. These controller parameters include both

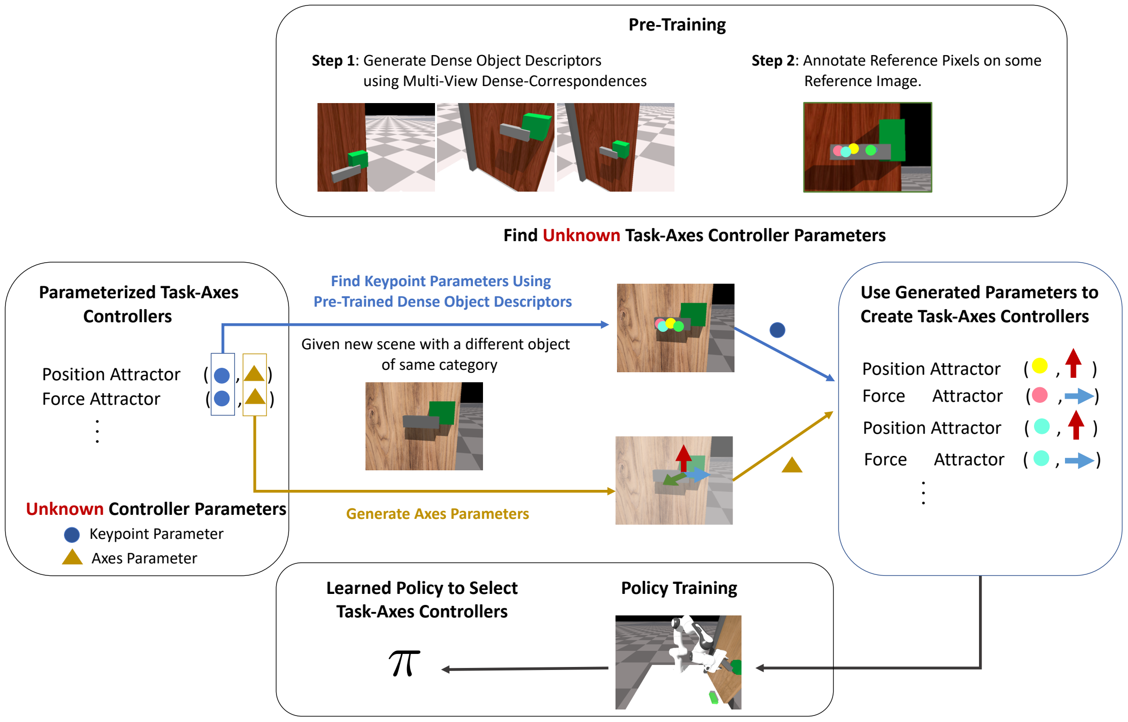

Pre-Training
Step 1: Generate Dense Object Descriptors using Multi-View Dense-Correspondences
Step 2: Annotate Reference Pixels on some Reference Image.
Find Unknown Task-Axes Controller Parameters
Parameterized Task-Axes Controllers
Position Attractor
Force Attractor
Unknown Controller Parameters
Keypoint Parameter
Axes Parameter
Find Keypoint Parameters Using Pre-Trained Dense Object Descriptors
Given new scene with a different object of same category
Generate Axes Parameters
Use Generated Parameters to Create Task-Axes Controllers
Position Attractor
Force Attractor
Position Attractor
Force Attractor
Learned Policy to Select Task-Axes Controllers
π
Policy Training

Figure 3.1: Overview of our proposed approach. We extend task-axes controllers to operate on visual input and use them to present a simple and generalizable approach for learning manipulation tasks.

the *position targets i.e.* 3D positions for relevant objects or other semantically meaningful points on the object such as edges or corners, as well as the *axes targets i.e.*, the axes along which the controller acts. Our aim in this work is to extend [213] to allow it to operate directly on visual input. More specifically, instead of using fixed heuristics to find the position-target parameters we would like to infer them directly from visual data. This is important since the previously used heuristics are often defined as functions of object parameters, and thus assume direct knowledge of an object's shape, size, and overall geometry, which might not be easily available in the real world. Additionally, instead of using heuristics to provide the axes-parameters we populate them automatically for each of the task-axes controllers. This results in an overall approach that requires minimal user input and allows the robot to learn complex manipulations tasks directly through interaction.

Our overall contributions in this paper include: 1) We marry two seemingly different approaches for generalization of manipulation tasks. Specifically, we show how multi-view dense correspondence learning can be effectively used with task-axes controllers, allowing for a simple and yet flexible approach for learning manipulation tasks. 2) We empirically validate the above approach on multiple manipulation tasks and show the generalization abilities of the resulting approach across both object (shape, size) and environment (lighting) properties.

3.2 Related Work

Task Frames: As noted in [213], task-axes controllers are closely related to the notion of 6D task frames [11, 12, 142]. Task Frames have been long been used in the robotics community. The early works of [11, 142, 181] formalized and used the notion of task-axes and constraint based task-frames (also referred to as constraint frames) for different manipulation tasks. Also, the authors of [11] noted the generalization ability that the geometric level of task frames provide by serving as a common middle ground between symbolic actions and motor control input. Since then task frames (often referred to as task-spaces) have found widespread use in robotics, for both planning and control [17, 100, 103, 138, 151, 155, 243].

However, only recently, methods have been proposed to learn to select the appropriate task frame for the given task [31, 103, 155, 177, 243]. All of these methods cast the task-frame selection problem as an imitation learning (IL) problem. Each of these methods advocate the use of some manually defined heuristic such as inter-trial variance between multiple demonstrations or the convergence behavior of demonstrations when viewed from different frames. In contrast to IL, [213] proposed an approach that uses Reinforcement Learning (RL) and learns to compose multiple task-axes based controllers to perform different manipulation tasks. The main motivation behind [213] is that many manipulation tasks often require multiple different subtasks to be performed in parallel. Each of these subtasks are accomplished by using different task-axes controllers and multiple task-axes controllers are combined hierarchically at each step to complete the overall manipulation task. Our work in this paper extends [213] to allow task-axes controller parameters to be inferred directly from visual input. As discussed previously, this is beneficial, since it avoids using heuristics, which often require direct knowledge of an object's pose, shape, size, and overall geometry. Another difference between our work and [213] is that we avoid using any prior knowledge to define target axes parameters. Example of such priors include – force controllers should be applied normal to surfaces and not across them. Instead, we would like the robot to learn such priors directly through interaction. This is important, since it allows us to create a simple method which requires minimal user input but can still learn to perform complex manipulation tasks.

Manipulation Skills can be represented in many different ways. Early work on manipulation skill representation used both geometric and mechanical features of the task to define manipulation primitives such as position, force primitives or visual servoing primitives [69, 153]. Contact states have also been used to define manipulation skills [64]. While early skills were often manually parameterized, recent skills have become more versatile utilizing learned parameters and preconditions [53, 81, 104, 211, 237, 247]. Some of the widely used skill representations include, Dynamic Movement Primitives (DMPs) [71, 111, 169], Neural Networks [87, 187, 198], Task-Parameterized GMMs (TP-GMMs) [24, 25]. In contrast to the above approaches, we do not propose a new skill representation but instead we utilize these different skill representations and learn to select the most appropriate skill-representation for the underlying task.

Hierarchical RL: Composing task-axes controllers for performing RL tasks is also related with hierarchical RL (HRL) [30, 86, 105, 238]. HRL uses the notion of options, which are temporally-extended actions, and learns to combine them to accomplish a given task. There has been a large body of work which aims to extract the underlying options [9, 135, 224, 233]. Many works aim to find options by using bottleneck states [146, 233], or policy sketches [217], while other works have also looked at the use of expert demonstrations [35, 108, 112, 214]. The use of task-axes controllers is different than the traditional use of options in RL since options are only composed temporally, while task-axes controllers are composed both hierarchically *i.e.* at each step as well as temporally.

Multi-view Correspondence Learning: Our approach also builds upon the recent work on using keypoints for manipulation learning [48, 49, 140]. Most closely related to our work is [140] which uses keypoints on objects to define a specific optimization problem for each task. Solving this optimization problem leads to a transformation in $SE(3)$ which is executed to perform the task. By contrast, we use the inferred keypoints to solve a temporally extended RL problem, wherein the robot learns by interaction. Another difference between the two approaches is that [140] manually annotate the relevant keypoints for each task and do not use self-supervised dense correspondence learning [49, 199].

3.3 Preliminaries: Skill Fundamentals

We use a compositional approach towards skill representation. Our skill representation is grounded in the geometry of the given task and its associated environment. We utilize a task-frame based formulation to utilize this geometric representation. Task-frame formalism (TFF) [21, 36, 113] allows us to associate 3D frames with task relevant objects in the environment. Instead of relying on full 3D task-frames we decompose each task-frame into a set of task-axes and only use the relevant axes for each task. This formulation is important in scenarios where frames or axes on different objects are required for control, *e.g.* in a wiping task we only need the surface normal of the surface being wiped while the target location is the marker to be wiped (Figure 3.3 Right).

Skill Representation: Each skill in our formulation consists of a set of one or more task-axes controllers (TACs). Overall, each task-axes controller (TAC) is parameterized by an object keypoint $(k \in K)$ that acts as the anchor for the controller, an end-effector keypoint $(e \in E)$ which indicates which part of the end-effector will interact with the object, a task-axes $(u \in U)$ that denotes the motion axes, and a controller with type $(c \in T)$ and corresponding parameters (θ) (*e.g.* DMP weights, or control-points for splines), which is used to generate the motion. Table 3.1 lists these elements and we describe each of them below. We use uppercase symbols K, E, U, T to denote sets of end-effector and object keypoints, axes, and controllers. Each element of these sets is indexed using lowercase symbols, *i.e.*, k, e, u, c respectively. Figure 3.2 visualizes how TACs are combined together to achieve skill representations.

3.3.1 Object and End-Effector Keypoints

We utilize keypoint-based representations to ground the task-axes controllers in the task environment. In this work, we assume a fixed set of object keypoints, denoted as K, which can be tracked in the environment. It is possible to predict these keypoints directly from the image input or the 3D point cloud representations. However, this often requires pre-training to learn features based on local object geometry [49, 199]. Further, in some scenarios using object-part centric approaches [39] may also be useful, *e.g.* grasping an airplane model from its tip. All of the above approaches can provide suitable keypoint representations for TACs.

TAC Element	Values
Object-Keypoint	Keypoints on object surface, center of object, object edges or corners
End-Effector Keypoints	End-effector center, Points on fingers, Palm points, Palm edges, Tool Keypoint
Task Axes	Global axes, Surface normals, Object axes, Linear axes to goal keypoints
Controller Types	Min-Jerk controllers, Force controllers, Alignment controllers, DMPs, Neural Networks

Table 3.1: Different elements of task-axes controllers along with the possible values each element can take.

In addition to the object keypoints we also utilize end-effector keypoints E that ground the end-effector interaction with the object. Figure 3.3 visualizes the palm end-effector keypoints. Overall, we use the end-effector center, finger keypoints, tool keypoint or palm keypoints. Although for most interactions the middle keypoint of the end-effector suffices, for many scenarios utilizing the fingers of the robot or the palm surface may also be relevant.

3.3.2 Task Axes

Each TAC is associated with some axis $u \in U$ along which it acts ($u \in \mathbb{R}^3$). While prior works often assume full 3D frames associated with each keypoint [52, 68], we decompose these frames into separate task axes. This is beneficial since in many scenarios only some axes of the full 3D reference frame are useful. For instance, in the peg insertion task or wiping tasks in [52] it is only the Z-axes of the hole or the surface normal of the board which are important, while the other axes can be more arbitrary *e.g.* any two axes in the plane normal to the hole will suffice for circular peg insertion. Based on this, we note that for most manipulation tasks a common set of axes can be used. These include the global X, Y, Z axes, the surface normals on objects, object-axes (which can be found from object-pose if known), linear axes from ee-keypoints to anchor keypoints (that are estimated as $k - e$, where k is an object keypoint and e is the end-effector keypoint), and joint-axes which can be used to define motion constraints.

3.3.3 Controller Types and Parameters

Each TAC is further associated with a feedback-based controller that generates the robot motion. These controllers can combine trajectory generators and feedback controllers together to generate the final robot motion. These TAC controllers can utilize the object and end-effector keypoints discussed above. However, they have their own set of parameters c_θ for motion generation. Table 3.1 lists the possible set of controllers.

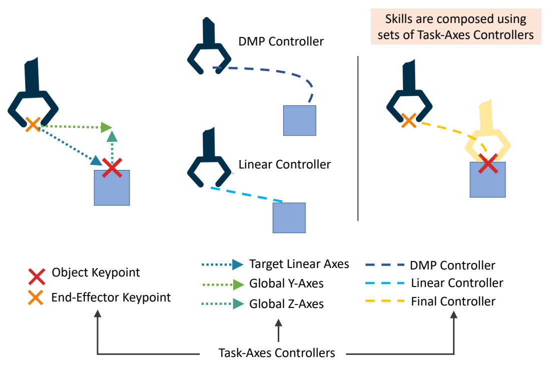


Figure 3.2: Skill composition using different TAC representations.

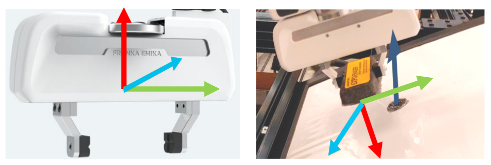


Figure 3.3: *Left:* Palm keypoint with an arbitrary frame. *Right:* Tool keypoint with EE frame and surface normal (black arrow) for black mark on the board. The other in-plane axes for the black mark are much more arbitrary.

Min-Jerk controllers are parameterized with the goal keypoint $k \in K$ and an offset $k_o \in \mathbb{R}^3$ with respect to these keypoints. Thus, the final target for the given end-effector keypoint is $k+k_o$, and these controllers act along the provided task-axes. They are further parameterized by the duration to reach the target.

Force controllers are parameterized by a desired force target $f_t \in [f_{\text{min}}, f_{\text{max}}]$ and act along the given task-axes.

Dynamic Movement Primitive based controllers are parameterized with goal location (*i.e.* the keypoint k and keypoint offset k_o). DMPs can be used to imitate arbitrary smooth motions by learning appropriate shape parameters $w \in \mathbb{R}^N$, (where N are the number of basis vectors) used to represent the forcing function [71, 111].

Alignment controllers are used to align some axes of the end-effector with some task-axes such as the surface normal of the whiteboard to wipe. These controllers can additionally be parameterized with (θ, ϕ) which specify the remaining 2-DOF for the 3-DOF orientation space.

Neural Network controllers can be parameterized with respect to goal locations (which are inferred from keypoints and offsets). The parameters for neural network controllers consist of all the weight and bias parameters in their layers. Given different TACs we compose them together using null-space projections similar to [213].

3.4 Learning Visual Controller Parameters

As discussed previously, [213] defines controller parameters such as x_c using heuristics *e.g.* using the middle of the door handle or the middle of the wall as target positions. Similarly, the axis parameters u were either set to the world axes or the object axes. However, each controller is only instantiated with the relevant axes, *i.e.*, some prior knowledge is provided by either specifying which target-axes is relevant for the current controller or which axes of the end-effector should possibly align with the known target axes. For instance, to clean a planar 2D surface we should only apply force control along the normal of the surface and not along the surface. Our main in this paper is to: 1) Extend task-axes controllers to operate directly on visual input, which avoids the usage of heuristics to specify the controller parameters, 2) Reduce the use of prior knowledge to specify relevant axes-parameters, which results in an approach that requires minimal user input and still can be used to solve complex manipulation tasks.

From the discussion in 3.3.3 we see that both position and force controllers are parameterized by a 3D position target and a 1-D target axes. Rotation controllers are additionally parameterized by the EE-axes used to orient with some target axes. We next explain our proposed methods to infer the relevant 3D position targets (keypoints) and target axes directly from visual inputs and interactions.

3.4.1 Keypoint Parameters

To infer keypoint parameters *i.e.*, the 3D position targets we use multi-view dense correspondence learning. Specifically, we use DenseObjectNets [48, 49] which learn dense object descriptors for each pixel from multi-view data in a completely self-supervised manner. Not only does this setup avoid the need of any expensive manual data-labeling procedure, prior works have shown that the learned object descriptors are quite robust in presence of mild occlusions and importantly, lead to category-level generalizations [48]. This category level generalization is extremely desirable for task-axes controllers since the controller position targets are often associated with semantic parts of the objects such as middle of the door handle to rotate it, or edge of a block to tumble it. Thus, category level generalization allows us to infer consistent controller targets irrespective of the object's size and position as well as variation in its shape and geometry.

To train dense object descriptors for our scenario, we use a small set of objects (≈ 10) relevant to each task family and learn dense descriptors on these objects. All of these objects belong to the same category, *e.g.*, for the door opening task we learn descriptors across doors with varying sizes of door handles as well as their locations on the door frame. Thus, the learned dense object descriptors should generalize to other novel objects that belong to the same category, *e.g.*, door handles with complex shapes. Figure 3.10 plots some learned object descriptors.

Given dense object descriptors, we extract the target keypoints for each controller by using a *reference* image from the dataset collected for training dense descriptors. This reference image is used as representative for the object category being manipulated. More specifically, we use this reference image to manually label some *reference pixels* on it. These reference pixels represent semantic information about the scene which is relevant for the task. For instance, for the door handle object we label pixels near the edge and middle of the door handle since these keypoints afford rotating the door handle compared to some position closer to the handle's rotation joint. Similarly for the block tumble task we label pixels near the edge instead of the middle of the block. Figure 3.9 shows example keypoints for the tasks considered in our current work. Each of these reference pixels is then used to create the desired position targets for both force and position controllers. To create position targets for a new scene which contains a novel object of the same category, we use the dense descriptors at the *reference pixels* and find pixels on the new object with the closest matching descriptor. These corresponding pixels should represent the same semantic information as the reference pixels *e.g.* close to the edge of the door handle. We set the 3D location of these corresponding as position targets for the task-axes controllers. This step is done before each training episode as well as while executing the learned policy at test time.

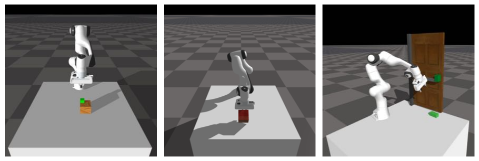


Figure 3.4: The three different tasks used to evaluate our visual controller parameter approach. From *left* to *right*: Button Press, Block Tumble, and Door Opening.

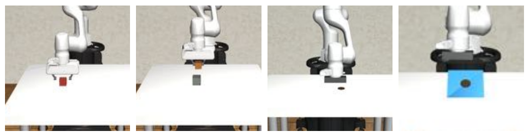


Figure 3.5: Robosuite [269] tasks used to evaluate the efficiency of using our structured skill representation approach. From *left* to *right* – Lift, Stack, Wipe, Wipe with slanted board.

3.4.2 Axes Parameters

Given keypoint parameters for each controller, we need to associate each keypoint with some set of axes. These axes are assigned as values to the axes-parameter defined above. To find this axes set we first find the relevant axes for the given task, which we refer to as the *candidate axes* set.

Although dense object descriptors allow us to extract relevant keypoints, these descriptors do not convey any information about the possible candidate axes values. In this work we use heuristics to extract the candidate axes. Specifically, as possible candidate axes we use both the global (world) and the object axes. Additionally, we also extract candidate axes by using the local geometry of the object around each inferred keypoint *e.g.*, using the axes normal to surface (*surface-normal*) or along the surface (*surface-tangent*).

Given the set of candidate axis, we need to associate each inferred keypoint parameter with some subset of this axis set. Instead of using user defined priors [213] or additional data (e.g. demonstrations [209]), Instead of using user-defined priors [213], in this part we use a combinatorial approach – we associate every keypoint with each of the axis in the candidate axes set. Since each keypoint-axes pair defines a controller, this combinatorial mix of keypoints and axes leads to large set of controllers for the robot to choose from (*i.e.* a larger action space). In the following sections, we empirically validate how this choice affects the performance of task-axes controllers for different manipulation tasks.

Figure 3.1 visualizes the pipeline of our overall approach. Our proposed approach is simple as it only requires user input to annotate reference pixels. An integral part of the above pipeline are the reusable task-axes controllers, *i.e.*, we use the same set of controllers across multiple tasks

3.5 Learning and Selecting Skills using Demonstrations

As we show in 3.6 using a combinatorial action space, as discussed above, results in poor sample complexity and makes real-world learning impossible. Further, instead of solely using task-axes controllers we would also like the agent to choose between different skill representations. To achieve these goals we propose to use a *small set of demonstrations* (≤ 5) to narrow down the possible set of valid skills for the given task, and provide a multi-modal space of skills for the agent to explore.

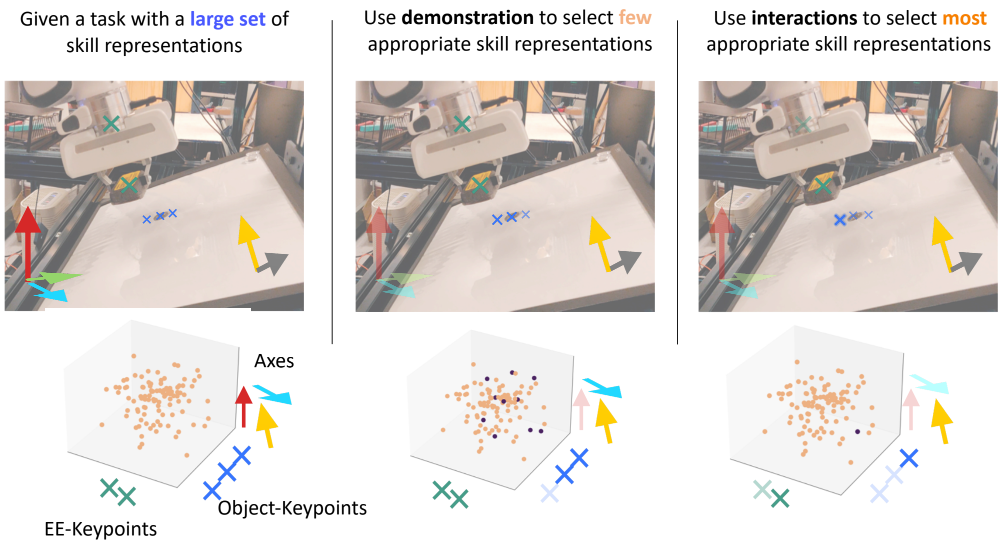

Given a task with a large set of skill representations
Use demonstration to select few appropriate skill representations
Use interactions to select most appropriate skill representations
Axes
Object-Keypoints
EE-Keypoints

Figure 3.6: We represent object and end-effector keypoints using × (object keypoints as ×, end-effector keypoints as ×) and axes as ↑. *Left*: Large set of skill representations. *Middle*: Use few demonstrations to select a small set of relevant skill representations. *Right:* Use RL to select the most appropriate skill representation for the task. The bottom plots show the likelihood values over different elements of our skill representation. *Bottom Left:* Initially, we have similar likelihoods for all skill representations. *Bottom Middle:* Using a demonstration we select skill representations with high likelihood (shown as black dots). *Bottom Right:* We use interactions to directly learn the most relevant skill representation (black dot) from this imitation reduced set.

3.5.1 Task Segmentation

Before learning appropriate TACs we segment a given task into multiple subtasks. This segmentation breaks down long horizon problems and is often necessary given the non-linearity around contact mode changes. We use change point detection to find contact mode transitions and use them to decompose the given task demonstration into multiple subtasks. For each subtask we now aim to find a subset of relevant TACs which are composed using null-space projections. TACs for all subtasks are then composed over time to accomplish the given task.

3.5.2 Computing Posteriors over Skill Representations

Given demonstrations for each subtask we would now like to infer a distribution over all sets of TACs to identify a small set of valid controllers that could be used to perform the task. For this we use a likelihood based approach. We denote by $q(k, e, u, c, c_\theta \in \mathbb{R}^N)$ some prior distribution over TACs. Given the demonstration data X we would like to find the posterior distribution over these parameters $p(k, e, u, c, c_\theta | X)$, where

$$p(k, e, u, c, c_\theta | X) \propto p(X | k, e, u, c, c_\theta) q(k, e, u, c, c_\theta). \tag{3.1}$$

Since the set of possible keypoints and controllers can be very large, we use some general priors to efficiently estimate the posterior distribution in (3.1). This set of priors is applicable across a large variety of manipulation tasks.

Proximity and Alignment priors: We first note that the geometric properties for the TACs, i.e, the object and end-effector keypoints and task axes can be separated from the feedback controllers. This is because given a set of values for these geometric properties multiple different controllers can be used to generate the robot motion. More specifically, the object keypoint parameter is used as an anchor for the controller the end-effector keypoint is used as the origin for the controller, and the task-axes projects the motion onto the appropriate direction. Given these parameters different feedback controllers can be used to generate the motion. We utilize this property of geometric properties to rewrite (3.1) as,

$$p(k, e, u, c, c_\theta|X) = p(c, c_\theta|X, k, e, u)p(k, e, u|X). \tag{3.2}$$

We estimate the second quantity in this product of posteriors $p(k, e, u|X)$ using proximity and alignment priors. A proximity prior allows us to find the most relevant keypoints based on the idea that relevant object and end-effector keypoints should be close to each other at the end of the interaction. Similarly, for the alignment prior we project the motion onto the relevant task axes and find proximity along this axes. Denoting the end-effector keypoint trajectory as $X^e_{1:T}$ (we use the subscript to denote time) we transform it to the appropriate object-keypoint and axes using

$$X^{e|k,u}_{1:T} = -\big(uu^T\big)(X^e_{1:T} - k), \tag{3.3}$$

where we use the notation $X^{e|k,u}_{1:T}$ to denote the trajectory for end-effector keypoint e with object-keypoint k as origin and along task-axes u.

Given this projected trajectory we use heuristics to estimate the likelihood for the given geometric parameters *i.e.*, $P(X|k, e, u)$. We use simple distance-based heuristic based on the last position of the demonstration trajectory,

$$p(X|k, e, u) = \exp\left(-\frac{||X^{e|k,u}_T - \hat{k}||_2}{\eta}\right) \tag{3.4}$$

where $\hat{k}$ is the 3D location for the given keypoint k and η is the temperature parameter. Using a low temperature value focuses on the closest keypoints alone while a larger value may use multiple keypoints. We note that other heuristics which look at the convergence behaviors of demonstrations can also be used [1, 103, 177].

Additionally, to estimate the full posterior $p(k, e, u|X)$ we can incorporate an informed prior $q(k, e, u)$,

$$p(k, e, u|X) = \frac{\exp\Big(-||X^{e|k,u}_T - \hat{k}||_2/\eta\Big) + q(k, e, u)}{\sum_{k',e',u'} \exp\Big(-||X^{e'|k',u'}_T - \hat{k}'||_2/\eta\Big) + q(k', e', u')}$$

These priors encapsulate task knowledge to guide the learning process. For instance, for prehensile manipulation a relevant prior is to use the middle EE-keypoint, while for tool-based tasks, when the hand is already grasping an object, the prior can focus on tool keypoints. Previous approaches to task-frame selection, and the more recent keypoint based approaches, use similar priors on end-effector keypoints [52].

3.5.3 Controller Selection and Learning

To estimate $p(c, c_\theta | X, k, e, u)$, we need to calculate the likelihood $p(X, k, e, u | c, c_\theta)$. For this we we use a set of criterions for the different controller types listed in Table 3.1. We note that as before we project the trajectory data onto the particular axes similar to (3.3). We use force-controller only in scenarios where the observed force in the projected force-data is beyond a threshold. We use a threshold of 2N in simulation and slightly larger 6N in real world given noisy force-torque measurements in real world. The force controller is parameterized by a force-value which is directly inferred from the demonstration.

For linear controllers (min-jerk) we use a simple linear regression on the non-zero velocity part of the trajectory to verify if the linear motion does not underfit the observed trajectory. We parameterize a linear controller with the time required to reach the target. On the other hand, DMP and Neural-Network (NN) controllers can fit arbitrary trajectories and thus we can always learn parameters for these controllers that correspond to the observed demonstration trajectories. However, given limited training data NN controllers can easily overfit to the trajectories and may not generalize well. We provide more details on controller criterions in the appendix on the project website.

3.6 Experimental Setup

With our experiments we aim to evaluate: 1) How well can we infer the keypoint controller parameters and use them to accomplish different manipulation tasks? 2) How well do the inferred task-axes parameters and the resulting policies generalize, *i.e.*, we would prefer to train our RL agent on as few object instances as possible and generalize to new objects of varying shapes and sizes.

Further for the second part of our work we use a single task demonstration to answer the following questions: 1) Can our proposed demonstration-based approach extract relevant skill information and discard irrelevant TACs using the demonstrations? 2) How efficiently can we use the reduced TACs for overall task completion? 3) Does our approach allow real-world learning on the robot?

3.6.1 Tasks

To evaluate our proposed approach we use 3 different manipulation tasks of increasing complexity – Button Press, Block Tumble, and Door Opening. Figure 3.4 visualizes the three different tasks.

Button Press: In this task, a 7-DoF Franka Panda arm is used to push down a button which is positioned on top of a box placed infront of the robot (Figure 3.4 left). Instead of using only one type of button object, we verify our approach on multiple objects with different shapes and sizes. These variations include

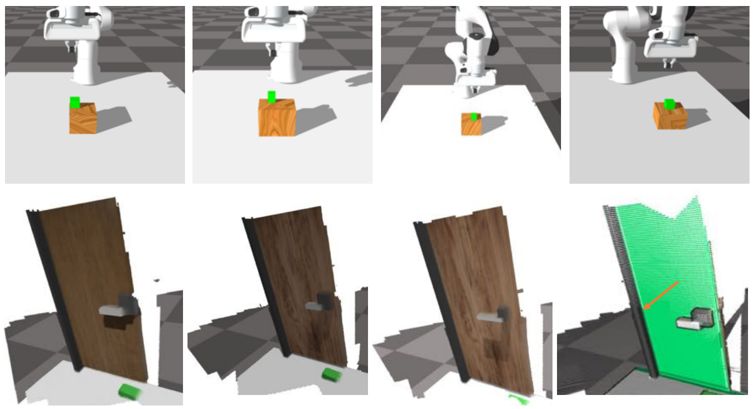


Figure 3.7: Example task variations used for training the Button Press and Door Open tasks.

the button position on top of the box object, sizes of both the button as well as its underlying box object. Figure 3.7 (top) visualizes some of these variations.

Block Tumble: In this task, a 7-DoF Franka Panda arm is required to tumble a block along a particular axis (Figure 3.4 middle). This task is particularly interesting since there exist multiple different ways to accomplish it. For instance, the robot can tumble the block by applying a downward force anywhere along its edge. To test generalization for this task, we vary the size of the block between 0.07m to 0.16m.

Door Opening: We also verify our approach on the door-opening task used in [213]. In this task, the Franka robot needs to open a door by first turning its door handle and then pulling the door beyond an opening threshold. In contrast to [213], which tests generalization by only varying the position of the door handle on the door frame, we vary both the size of the door handle as well as the location of the door handle on the door. Additionally, we also change the door shape and evaluate the proposed approach for both cuboidal, cylindrical and more complex door handle shapes. See Figure 3.7,3.11 or attached video (project page) for reference.

To evaluate how the use of demonstrations impacts our approach we use three different tasks from robosuite [269] – *Lift*, *Stack*, and *Wipe* (Figure 3.5). We use [269] tasks since they provide infrastructure to collect demonstrations in simulation.

The robosuite tasks we use are of varying difficulty and evaluate different aspects of our approach. For the Lift task we do not make any changes to the robosuite env. While for Stack we make one change, *i.e.*, we ensure that the blocks to be stacked will have a minimum distance of $3cm$ between them. This ensures that the demonstration trajectory environment and the train environments align with each other. Finally, for the Wipe task we make two changes. First, we reduce the number of markers to only 1, this makes it easy for us to test our approach in the real world, since we can use a simple vision system to detect this marker. Second, we stochastically add a slanted board (Figure 3.5 right) and place the marker on this board instead of the table surface. This requires the agent to better generalize the wiping task.

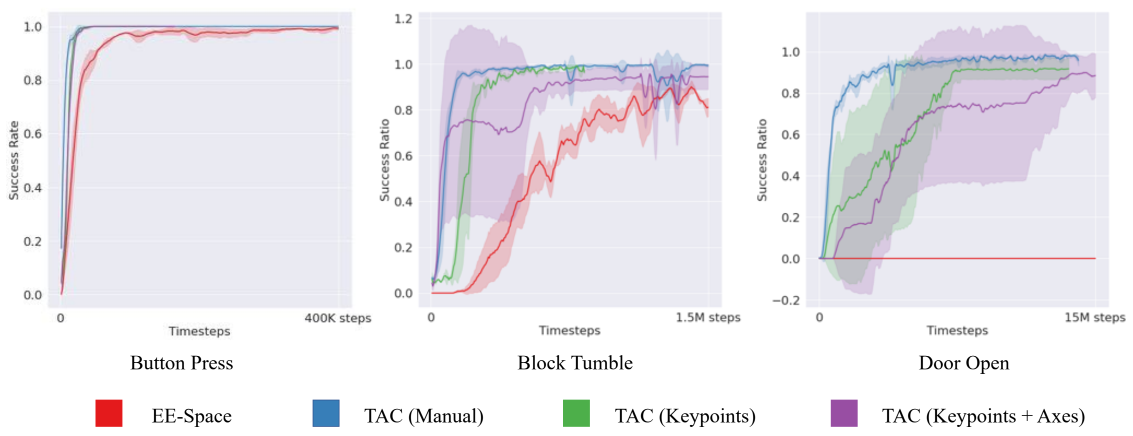


Figure 3.8: Task Success Rate for all three different environment. The dark line shows the mean success-rate while the shaded region plots the std across 5 seeds.

3.6.2 Compared Approaches

To evaluate our proposed approach we compare multiple different methods. In the below sections we refer to task-axes controllers using the shorthand TAC.

1. **EE-Space:** We verify the utility of the structured action space provided by object-centric task-axes controllers by comparing against an approach that directly controls the robot via end-effector delta targets.

2. **TAC (Manual)** We also evaluate our approach against manually specified controller parameters. We note that we do not aim to use the *smallest* possible number of controllers and their parameters. Instead, we specify the controllers and their parameters only to provide some useful priors for overall task learning.

3. **TAC (Keypoints):** We evaluate one version of our proposed approach in which we only infer the keypoints *i.e.* the target positions for each position or force controller. We reuse the axes specified for TAC (Manual) with the inferred keypoints.

4. **TAC (Keypoints+Axes):** We evaluate our proposed approach wherein both the keypoints as well as the axes parameters are inferred for each scene.

Table 3.2 shows the number of keypoints and axes parameters inferred for each of the tasks. We note that there exist controllers which do not have any axes associated with them such as the error axis controllers [213].

To evaluate how the use of different skill representations affects our demonstration guided approach we use another set of baselines. These baselines include commonly used skill representations.

1. **BC with DMPs:** We compare our approach against a behavior-cloning (BC) baseline. Given that we only use one expert trajectory we use DMPs for BC instead of neural networks, since the latter would require a much larger number of expert trajectories for generalization.

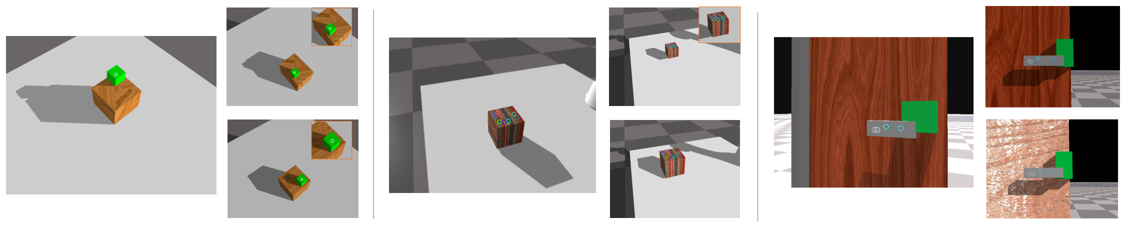


Figure 3.9: Visualization for *reference* images and pixels (left image in each column) and corresponding pixels predicted using learned descriptors. For door open, one reference pixel is closer to door joint to show that our approach learns to not use it.

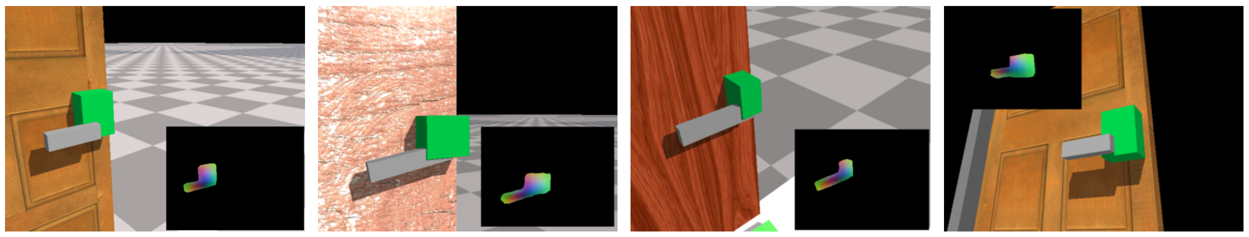


Figure 3.10: Dense object descriptor results for Door Open Task.

2. **BC + RL with DMPs:** In addition to behavior cloned DMPs, we also compare against an approach in which we finetune the learned DMPs with RL.

3. **RL:** We also compare against an RL based approach. For this, we tried using both on-policy *i.e.* Proximal Policy Optimization (PPO) [201] as well as off-policy Soft-Actor Critic (SAC) [57] approaches. However, the PPO based approach did not make any progress within sufficient amount of time, hence we show results only for SAC. Additionally, for SAC we tried using the single expert demonstration by adding it to the replay buffer, however this did not yield any advantage.

4. **TACs:** We evaluate our proposed approach wherein all TAC elements including keypoints, ee-keypoints, axes, controller types are first selected from demonstration using IL and then refined using RL.

Task	TAC (Manual)	TAC (Keypoints+Axes)	Keypoints
Button Press	5	14	2
Block Tumble	10	40	10
Door Open	8	51	4

Table 3.2: Controller Statistics for each of the tasks.

Metrics: We compare all approaches using the success ratio metric. We report the success ratios of the learned policies separately for both train and test environment configurations. For each task we use 4 train configurations while we test on 15 different configurations for each environment.

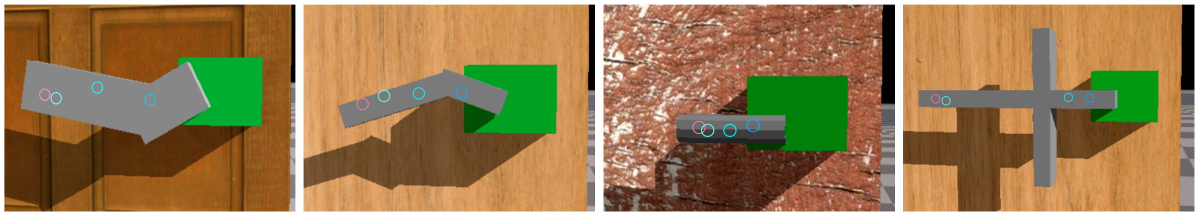


Figure 3.11: *Qualitative Results* we show that our learned control policy although not trained on any of the above models does successfully transfer to them (see results in project-page).

Env	EE-Space	TAC (Manual)*	TAC (Keypoints+Axes)
Button Press	0.98 (0.01)	1.0 (0.0)	1.0 (0.0)
Block Tumble	0.487 (0.26)	0.96 (0.03)	0.932 (0.04)
Door Opening	0.0 (0.0)	0.97 (0.01)	0.94 (0.05)

Table 3.3: Mean (std) for each method on all three different manipulation tasks. Each approach was verified on 15 different environment variations.

3.6.3 Metrics and Training

For our visual task-axes controller experiments we train all methods using Proximal Policy Optimization (PPO) [201] using stable-baselines [65]. All results are run and reported for 5 different seeds (we do not use any top-K seeds criterion). We use same hyper-parameters as used in the door-opening task of [213].

For the RL baseline to select appropriate skill representation experiments we use the code provided by the robosuite-benchmark. We use the same hyperparameter settings provided by them and run their code on the slightly modified environments.

We use success ratio as the metric to quantitatively compare the approaches, since reward as a metric may not be consistent across different baselines due to their different horizons and overall steps. Also importantly, we plot the success-ratio metric against the trajectory steps instead of environment steps. This is because in trajectory based approaches (such as DMPs or ours) a single environment step corresponds to multiple simulator steps. Thus, we also divide the RL steps by the maximum number of trajectory steps.

3.7 Results

3.7.1 Learning Visual Controller Parameters

Figure 3.8 plots success ratio for all approaches across each of the three different tasks. As seen above, we observe that for the simplest task *i.e.* button press, all methods are able to learn the task quickly. Also, since the underlying task is not complex, each method has little variance across multiple seeds. For the Block Tumble task (Figure 3.8 middle), although all methods perform well on the training task, methods that use task-axes controllers are much more sample efficient. This is true even when we use a much larger set of controllers *i.e.* the TAC (Keypoints + Axes) approach. This is because most of the keypoints used in the task (Figure 3.9) can be used to accomplish the task. Also, since there is only one axes along which the block needs to be flipped, the robot is quickly able to find this relevant axes using the provided dense rewards. Thus, even with a much larger action space our approach of inferring the keypoints and axes performs similarly, when compared to their manual specification.

Figure 3.8 (right) plots the success ratios for the door opening task. From the above figure, we observe that the EE-space is unable to solve the overall task. Similar results were also observed in [213]. The main reason for this failure is the overall task complexity, especially since task completion requires many different subtasks (reaching, grasping, turning the handle and pulling it back) to be performed in sequence. On the other hand, we observe that all methods that utilize task-axes controllers are able to learn to perform the task. However, in contrast to the previous tasks, our proposed approach, TAC (Keypoints+Axes), does require more samples compared to when we manually provide these parameters, TAC (Manual). We also observe that only inferring the keypoints and not the axes (TAC - Keypoints) is still quite sample efficient. This indicates that the agent in TAC Keypoints+Axes does spend some initial time exploring different axes which can be used to accomplish the task. This is possible because there exist multiple ways to grasp the handle. For instance, it is possible to grasp the handle both along the vertical as well as the horizontal axes. However, using the vertical axes is not robust, since it can easily collide with the door frame. Thus, as a large number of actions are not particularly useful for the task, the agent will have to interact and learn the most suitable and robust ways to achieve it.

Generalization Results

Table 3.3 shows the generalization performance for three different methods. This generalization performance was recorded on 15 different environment settings with varying object sizes and shapes. We note that for TAC (Manual) we only used primitive shapes (cuboids and cylinders) since we need to manually provide keypoint parameters. By contrast, for TAC (Keypoints+Axes), we directly use the visual input to infer the relevant keypoint parameters. See Figure 3.11 for test configurations used for TAC (Keypoints+Axes), for door open task. As seen in Table 3.3, both methods that use task-axes controllers are able to generalize quite well across all of the tasks. On the other hand, the EE action-space provides good generalization capabilities only for the simplest task (Button Press). While even for the moderately complex task of Block Tumble its performance reduces significantly. One reason for this drop in performance is the lack of any inductive bias in the EE-space, and since we train on a small set of objects only, the EE-space policy fails to generalize to larger changes in object variations. Figure 3.11 shows some objects with complex underlying geometry that were never used either for dense descriptor or policy training. While TAC (Manual) cannot be applied to such objects, we show in the attached video results that our method is successfully able to zero-shot generalize to such large variations as well.

Qualitative Results

In addition to the above quantitative results, we also show some qualitative results for the learned descriptors and the inferred keypoints. Figure 3.9 plots the keypoints used for each of the tasks. The left image in each column is the *reference* image with the annotated *reference* keypoints. While the right column shows scenes with two *different* objects used in the test set to evaluate the learned policies. We visualize objects at same scale to show their original sizes. Additionally, in Figure 3.10 we plot the learned descriptors for the door opening task. As seen above, the learned descriptors are able to approximately cluster semantically

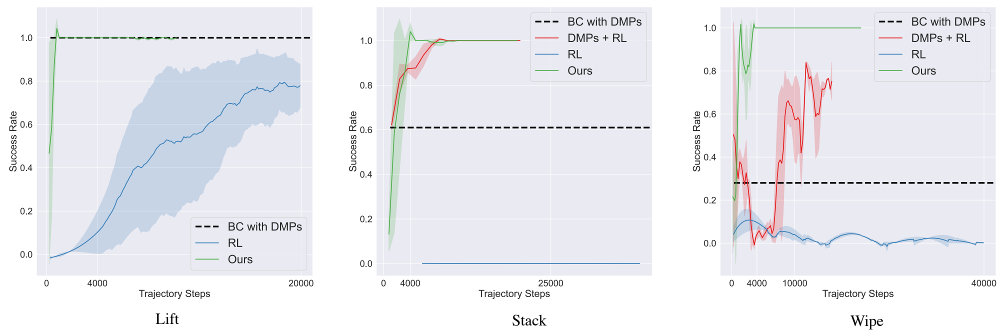


Figure 3.12: Success Ratio for the different tasks – Lift, Stack, and Wipe. All tasks in simulation are run for 4 seeds.

meaningful regions together. For instance, the part of door handle close to its rotation joint, the middle and end of the door handles, as well as the right end of the hinge are all well estimated. While in Figure 3.11 we see that the reference pixels are also able to generalize to objects with very different shapes and geometry. We also show qualitative results for these samples in our video results. This is not surprising since as long as the above keypoints afford grasping and rotating the door handle, the underlying task-axes controllers should be able to generalize. See video results for all tasks and supplementary material at `https://sites.google.com/view/robotic-manip-task-axes-ctrlrs`.

3.7.2 Results - Efficiently Learning Skill Representations using Demonstrations

Given the reduced set of skill representations we now discuss the results of choosing (and refining) the most appropriate controllers using RL. Figure 3.12 plots the success ratio for each of the environment across all approaches. This success ratio is obtained using the deterministic policy by running it for 50 episodes intermittently during training. Also, as noted in Section-3.6.3 the X-axes in Figure 3.12 refer to the trajectory steps and not the environment steps.

From Figure 3.12 we can see that for the simplest task, Lift, the DMPs learned from demonstrations directly suffice to complete the task. By comparison, for our approach, since there exists multiple similar TACs initially, a small amount of exploration is required to arrive at the optimal solution. For the Stack task (Figure 3.12 Middle)), learned DMPs achieve around 60% success rate and do not generalize perfectly to all test scenarios. These scenarios include when the objects are placed in very different configurations than in the provided expert demonstrations and consequently the robot often ends up hitting the other object instead of moving above it. We provide qualitative results of this failure on our website. Alternatively, using these learned DMPs with our TAC skills or performing RL with these DMPs does learn to solve the task quickly and robustly. However, our approach is slightly more sample efficient since the high-level policy chooses to use the linear min-jerk controller for placing instead of a DMP controller for the XY axes. This linear controller is only parameterized by its velocity and hence is faster to optimize than a 10-dimensional DMP. This result signifies the advantage of our approach wherein the higher level policy is able to choose the most appropriate controller to optimize.

Finally, the Wipe task Figure 3.12 (Right) requires a more fine-grained contact interaction than previous tasks. We find that the learned DMPs perform comparatively worse (around 30%). This is because a

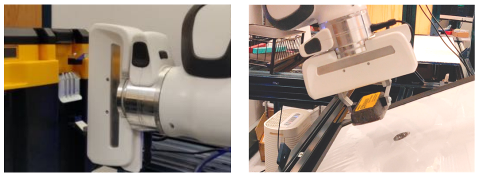


Figure 3.13: Real world tasks Open-Toolbox and Wipe.

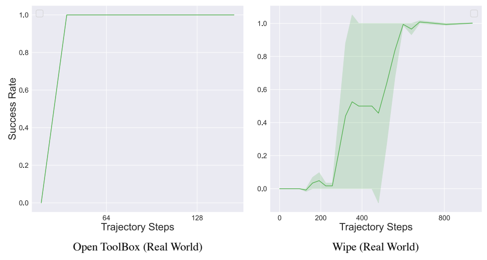


Figure 3.14: Success Ratio for real world tasks, run on 2 seeds.

sufficient amount of downward force needs to be generated to wipe the marker which is hard to learn from position trajectories alone. Training these DMPs with RL we find that although they can complete the task, the resultant policy can be very unsafe (see video on project website). Hence, we add an additional constraint to force the RL trajectory to have the final EE-location within a safe threshold $(0.12m, 0.04m, 0.04m)$ of the final demonstration location. Even with this change the DMP controllers are still less sample efficient and plateau early in comparison to our approach, which instead uses a force controller to generate the appropriate amount of force to wipe the marker.

3.7.3 Real World Results

We also evaluate our approach in the real-world on two different tasks – Open ToolBox and Wipe (Figure 3.13). We use these task to show that our approach can learn a policy from scratch directly in the real world (given a *single* demonstration).

Setup: Figure 3.13 shows out task setups. For the Open-ToolBox task we use a common toolbox and learn to open its locks. For Wipe, we use a setup similar to our simulation, wherein a single black mark needs to be erased. We use OpenCV's blob detector to get the marker positions. For both tasks we change the object orientations i.e. the toolbox and whiteboard orientations arbitrarily and use the covariance of point cloud around the keypoint positions to get their surface normal. We do not utilize any other instrumentation for state-estimation. We use the open-source FrankaPy [263] to run our code, and and use the same set of controllers as used in simulation.

We use one demonstration to reduce the initial set of TACs. We do not use DMP controllers during the interaction with the toolbox or whiteboard. We do this purely for safety reasons as is evident from the unsafe nature of updating learned DMPs for Wipe in simulation. During RL training we provide a reward of 1 to the agent for reaching close to the target object, and an additional reward of 4 for completing the task. Additionally, for the open-toolbox task we also provide an additional reward of 1 if the agent correctly aligns with the toolbox. We do this to evaluate how much of an effect dense rewards have on the learning process. In all other scenarios for both tasks we provide 0 reward.

Figure 3.14 plots the success ratio for both tasks, using a deterministic policy which runs for 5 episodes intermittently during training. As seen above, for both tasks we are quickly able to learn the appropriate controller sequence and parameter values in real world settings. However, the robot learns the Open-ToolBox task much faster than the Wipe task. This is because of two reasons: First, for the former we provide better shaped rewards (additional reward for alignment with tool lock). Second, the low level parameters inferred from the demonstrations for this task (e.g. force parameter) is sufficient for task completion, hence the agent requires less exploration to successfully complete the task.

While we do not observe any intra-seed variance for the Open-ToolBox task there exists some variance for Wipe. This is due to the high level policy (for one seed) not using the appropriate alignment and force-controller for performing the wiping task. Instead, initially, the policy incorrectly aligns with the whiteboard, and once the correct alignment is learned the policy only chooses the force controller after more exploration. However, once learned both of the policies are able to accomplish the task successfully.

3.8 Conclusion

Our work in this paper leads to a naturally modular architecture which separates perceptual learning from the control policy, *i.e.*, although the control policy acts on the perceptual input, both of the models are trained separately. This stream of work is similar to the recently proposed [140], wherein the predicted keypoints are used to solve an optimization problem whose output is used to perform the task. Despite differences between both approaches, they both lead to an improved interpretability of the learned models. This is a consequence of semantic keypoints in [140], while for our work this is a consequence of both semantic keypoints as well as task-axes controllers which operate on semantic inputs. In addition to interpretability, a modular architecture also implies that given the same control policy, we can retrain the perceptual network on a completely different set of objects (*e.g.* different geometries) and still learn to perform the task as long as the semantic keypoints for both of these objects provide the requisite affordances such as grasping, turning, pushing. This is in contrast to end-to-end DeepRL approaches where the perceptual network is closely tied with the control policy and it is not possible to update one without updating the other.

4 Planning with Learned Skill Effect Models for Lifelong Robotic Manipulation

This chapter is based on [Liang, Sharma, LaGrassa, Vats, Saxena, and Kroemer, 130].

Abstract: *Robots deployed in many real-world settings need to be able to acquire new skills and solve new tasks over time. Prior works on planning with skills often make assumptions on the structure of skills and tasks, such as subgoal skills, shared skill implementations, or task-specific plan skeletons, which limit adaptation to new skills and tasks. By contrast, we propose doing task planning by jointly searching in the space of parameterized skills using high-level skill effect models learned in simulation. We use an iterative training procedure to efficiently generate relevant data to train such models. Our approach allows flexible skill parameterizations and task specifications to facilitate lifelong learning in general-purpose domains. Experiments demonstrate the ability of our planner to integrate new skills in a lifelong manner, finding new task strategies with lower costs in both train and test tasks. We additionally show that our method can transfer to the real world without further fine-tuning.*

4.1 Introduction

Lifelong-learning robots need to be able to plan with new skills and for new tasks over time [241]. For example, a home robot may initially have skills to rinse dishes and place them individually on a rack. Later, the robot might obtain a new skill of operating a dishwasher. Now the robot can plan to either wash the dishes one by one or use the dishwasher depending on the costs of each skill and the number of dishes to be cleaned. In other words, robots need to be able to obtain and use new skills over time to either adapt to new scenarios, solve new tasks, or to improve performance on existing tasks. Otherwise, the robot engineer would need to account for all potential tasks and strategies the robot can use before deployment. As such, we propose a task planning system that can efficiently incorporate new skills and plan for new tasks in a lifelong robot manipulation setting.

To create such a versatile manipulation system, we use parameterized skills that can be adapted to different scenarios by selecting suitable parameter values. We identify three properties of skills that are important to support in this context: 1) skills can have different implementations, 2) skills can have different parameters which can take discrete, continuous, or mixed values, and 3) skill parameters may or may not correspond to subgoals. Property one means the skills can be implemented in a variety of manners, e.g., hard-coded, learned without models, or optimized with models. This requires relaxing the assumptions placed on the

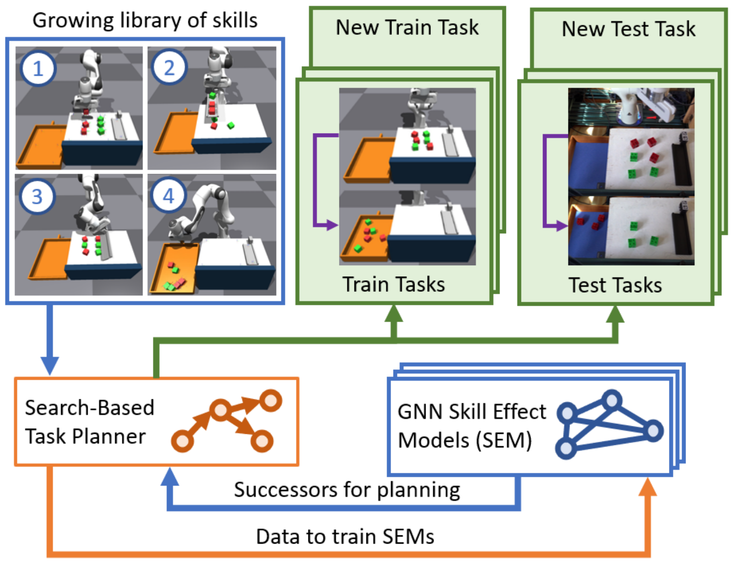


Figure 4.1: Overview of the proposed search-based task planning framework with learned skill effect models (SEMs) for lifelong robotic manipulation. New skills and training tasks can be added incrementally. We collect skill effects data by running the planner using all skills on all training tasks in simulation. The collected data is used to train GNN SEMs for new skills or fine-tune models of existing skills. Learned models predict both the terminal state and cost of skill executions. The planner can use SEMs to plan low-cost paths on test tasks in the real world. This approach supports planning 1) with a set of differently parameterized skills that can grow over time and 2) for test tasks unseen during training.

skill structures made in previous works, such as implementing all skills with the same skill-conditioning embedding space [125, 134, 206, 252]. Property two requires the task planner to not assume any fixed structure for skill parameters. Unlike previous works [137, 161], each skill can utilize a different number of parameters, and these parameters can be a mix of discrete and continuous values. Property three means that instead of chaining together skill subgoals, the planner needs to reason about the effects of the skills for different parameter values. For example, the home robot may need to predict how clean a plate is for different rinsing durations.

Planning for new tasks requires the planner to be flexible about the structure of task specifications. One way to do this is by using either goal condition functions or goal distributions [32], instead of shared representations like task embeddings [162] or specific goal states [70, 125, 161, 232]. Using predefined task representations limits the type of tasks a robot can do, and using learned task embeddings may require fine-tuning on new tasks. Only having a goal condition function also makes it more difficult to represent a task as an input to a general value or policy function implemented using a function approximator.

To satisfy the skill and task requirements for the lifelong manipulation planning problem, we propose a task planning system that performs **search-based planning with learned effects of parameterized skills**. Search-based methods directly plan in the space of skill-parameter tuples. A key advantage of search-based planning methods is they can use skills regardless of parameter choices or implementation details, and only need a general goal condition check to evaluate task completion.

To efficiently use search-based planning methods for task planning, we propose to learn skill effect models (SEMs). SEMs are learned instead of hardcoded or simulated, since manually engineering models is not scalable for complex skills and simulations are too expensive to perform online during planning. Every skill has its own SEM that predicts the terminal state and costs of a skill execution given a start state and skill parameters. We interleave training SEMs with generating training data by running the planner with the learned SEMs on a set of training tasks. Our data collection method efficiently collects skill execution data relevant for planning, and supports the addition of new skills and tasks over time. The planner uses the SEMs to plan for existing tasks with different initial states, as well as new test tasks.

Our contributions are 1) a search-based task planning framework with learned skill-effect models that 2) relaxes assumptions of skill and task representations in prior works; skill effect models are learned with 3) an iterative data collection scheme that efficiently collects relevant training data, and together they enable 4) planning with new skills and tasks in a lifelong manner. Please see supplementary materials, with additional results and experiment videos, at `https://sites.google.com/view/sem-for-lifelong-manipulation`.

4.2 Related Works

Subgoal skills. Many prior works approached planning with skills with the subgoal skill assumption. The successful execution of a subgoal skill always results in the same state or a state that satisfies the same preconditions of all skills, regardless of where the skill began in its initiation set [104]. As such, the skill effects are always known, and such approaches instead focus on learning preconditions [161] of goal-conditioned policies, efficiently finding parameters that satisfy preconditions [137, 222], or learning feasible skill sequences [70]. While subgoal planning is powerful, it limits the types of skills the robot can use.

Non-subgoal skills. For works that plan with non-subgoal skills, many represent the skill policy as a neural network that takes as input both the state and an embedding that defines the skill. This can be viewed as planning with one parameterized skill or a class of non-parameterized skills, each defined by a different embedding. Such skills can be discovered by experience in the real world [134] and in learned models [206, 252], or learned from demonstrations [125]. Planning with these skills is typically done via Model Predictive Control (MPC), where a short sequence of continuous skill embeddings is optimized, and replanning occurs after every skill execution. While these approaches do not assume subgoal skills, they require skills to share the same implementation and space of conditioning embeddings, and MPC-style planning cannot easily support planning with multiple skills with different parameter representations [125, 134, 161, 252].

Planning with parameterized skills. To jointly plan sequences of different skills and parameters, works have proposed a two-stage approach, where the planner first chooses the skills, then optimizes skill parameters [168, 222, 254]. Unlike directly searching with skills and parameters, it is difficult for two-stage approaches to give guarantees on solution quality. Some also require hardcoded or learned plan skeletons [222, 254], which limits the planner's applicability to new tasks.

Instead of planning, an alternative approach is to learn to solve Markov Decision Processes (MDPs) with parameterized skills [60, 143, 253]. However, learning value or policy functions typically requires a fixed representation for function approximators, so these methods cannot easily adapt to new skills and skills with parameters with different dimensions or modalities (e.g. mixed continuous and discrete). Doing so for search-based planning can be done by directly appending new skills when expanding a node for successors.

Obtaining skill effects. Many prior works used simulated skill outcomes during planning [98, 99, 168, 230]. This can be prohibitively expensive to perform online, depending on the complexity of simulation and the duration of each skill. To avoid simulation rollouts, works have used hardcoded analytical [10, 22] or symbolic [44, 88, 89] skill effect models. Manually engineering such models may not always be feasible, and they do not easily scale to changes in skills, dynamics, and tasks. Although symbolic models can be automatically learned [6, 104, 235, 242, 247], these approaches also make the subgoal skill assumption. By contrast, our method, which learns skill effect models in continuous states without relying on symbols, can plan with both subgoal skills as well as skills that do not share this property.

The works most closely related to ours are [254] and [247]. In [254], the authors jointly train latent dynamics, latent preconditions, and parameter samplers for hardcoded skills and a model that proposes plan skeletons. Planning is done MPC-style by optimizing skill parameters with the fixed plan skeleton. Although this approach does not assume subgoal skills and supports skills with different parameters, learning task-specific plan skeletons and skill parameter samplers makes it difficult to use for new tasks without finetuning. The method in [247] learns to efficiently sample skill parameters that satisfy preconditions. Task planning is done using PDDLStream [54], which supports adding new skills and tasks. Though this approach does not use subgoal parameters, the desired skill outcomes are narrow and predefined, and the learned parameter sampler aims to achieve these predefined effects. As such, the method shares the limitations of works with subgoal skills, where the skill-level transition model is not learned but predefined as the subgoals.

4.3 Task Planning with Learned Skill Effect Models

The proposed method consists of two main components - learning skill effect models (SEMs) for parameterized skills and using SEMs in search-based task planning. These two components are interleaved together - we run the planner on a set of training tasks using SEMs to generate data, which is used iteratively to further train the SEMs. New skills and training tasks can be added to the pipeline because the planner and the SEMs do not assume particular implementations of skills and tasks. The planner can also directly apply the learned SEMs to solve test tasks. See overview in Figure 4.1.

4.3.1 Skill Planning Problem Formulation

Parameterized skills. Central to our approach is the options formulation of skills [104, 238]. Denote a parameterized skill as o with parameters $\theta \in \Theta$. Parameters are skill-specific and may contain subgoal information such as the target object pose for a pick-and-place skill. We assume a fully observable state $x \in \mathcal{X}$ that contains all information necessary for task planning, cost evaluations, and skill executions. We define the low-level action $u \in \mathcal{U}$ as the command sent to the robot by a low-level controller shared by all skills (e.g. torque).

In our formulation, a parameterized skill o contains the following 5 elements: an initiation set (precondition) $\mathcal{I}_o(x, \theta) \rightarrow \{0, 1\}$, a parameter generator that samples valid parameters from a distribution $p_o(\theta|\mathcal{I}(x, \theta_i) = 1)$, a policy $\pi_o(x) \rightarrow u$, a termination condition $\beta_o(x, \theta, t) \rightarrow \{0, 1\}$, and the skill effects $f_o(x_t, \theta) \rightarrow x_{t+T}$, where T is the time it took for the skill to terminate. To execute skill o at state x with parameters θ, we first check if (x, θ) satisfies the preconditions $\mathcal{I}_o$. If it does, then we run the skill's policy π_o until the termination condition is satisfied. We assume that the preconditions, parameter generator, policy, and termination conditions are given, and the skill effects are unknown but can be obtained by simulating the policy. To enable reasonable planning speeds, the SEMs learn to predict these skill-level transitions.

To justify the assumption of given skill preconditions, we note that our preconditions are broader than ones in prior works and consequently can be easily manually defined. Preconditions in many prior works, especially ones that use subgoal skills, are only satisfied when a specific outcome is reached, so they may require learning sophisticated functions to classify which (state, parameter) tuple lead to the intended outcome [247, 254]. By contrast, because we allow non-subgoal skills, our preconditions are satisfied if skill execution leads to any non-trivial and potentially desirable outcome. For example, for a table sweeping skill, the preconditions are satisfied as long as the robot sweeps something, instead of requiring sweeping specific objects into specific target regions. Due to the broad and simple nature of our more flexible preconditions, we argue it is reasonable to assume they are given.

Task planning of skills and parameters. Before specifying tasks, we first define a background, task-agnostic cost $c(x_t, u_t) \geq 0$ that should be minimized for all tasks. This cost is accumulated at each step of skill o execution, so the total skill cost is $c_o = \sum_{t=0}^{T} c(x_t, u_t)$. A task is specified by a goal condition $\mathcal{G}(x) \rightarrow \{0, 1\}$ that classifies whether or not a state achieves the task. We denote a sequence of skills, parameters, and their incurred states as a path $P = (x_0, o_0, \theta_0, x_1, \ldots x_n, o_n, \theta_n, x_{n+1}, \ldots, x_N)$, where N is the number of skill executions, and the subscripts indicate the nth skill in the sequence (not time). We assume the environment dynamics and skill policies are deterministic. The task planning problem is to find a path P such that the goal condition is satisfied at the end of the last skill, but not sooner, and the sequence of skill executions is feasible and valid. See equation 4.1.

$$\begin{aligned} \min_P \quad & \sum_{n=0}^{N-1} c_{o_n} \\ \text{s.t.} \quad & \mathcal{G}(x_N) = 1 \\ \forall n \in [0, N-1], & \mathcal{G}(x_n) = 0 \\ \mathcal{I}_{o_n}(x_n, \theta_n) = 1, & f_{o_n}(x_n, \theta_n) = x_{n+1} \end{aligned} \tag{4.1}$$

Note that θ, $\mathcal{I}_o$, and f_o are all skill-specific, so with M types of skills, there are M different parameter spaces, preconditions, and skill effects.

4.3.2 Learning Skill Effect Models (SEMs)

Defining SEMs for manipulation skills. We learn a separate SEM for each skill, which takes as input the current state x_t and a skill parameter θ. The SEM predicts the terminal state x_{t+T} reached by the skill when it is executed from x_t using θ and the total skill execution cost c_o. We assume SEMs are queried only with state and parameter tuples that satisfy the precondition. Because we focus on the robot manipulation domain, we assume the state space $\mathcal{X}$ can be decomposed into a list of object-centric features that describe discrete objects or robots in the scene.

We represent SEMs using Graph Neural Networks (GNNs), because their inductive bias can efficiently model interactions among entities through message passing, encode order-invariance, and support different numbers of nodes and edges during training and testing [14, 74, 93, 239]. Each node in the SEM GNN corresponds to an object in the scene and contains features relevant to that object from the state x. We denote these object features as $s_k \in \mathbb{R}^S$, where k denotes the kth object in the scene. Because a skill may directly affect multiple objects, each node also contains the skill parameters θ as additional node features. The full node feature is the concatenation of $[s_k, \theta]$. There are no edge features. The network makes one node-level prediction, the change in object features Δs_k, and one graph-level prediction, the total skill execution cost c_o. As SEMs make long-term predictions about the entire skill execution, the graph is fully connected to allow all objects to interact with each other, not just objects that are initially nearby. The loss function to train SEMs for a single step of skill execution prediction is $\mathcal{L} = \lambda_c \|c_o - \hat{c_o}\|_2^2 + \frac{\lambda_s}{K} \sum_{k=1}^{K} \|\Delta s_k - \hat{\Delta s_k}\|_2^2$. The hat notation denotes predicted quantities, and the λs are positive scalars that tune the relative weights between the loss terms. The GNN is implemented with PyTorch Geometric [46].

SEMs enable efficient planning of diverse parameterized skills, as well as two additional benefits. First, because the model is on the skill-level, not action-level, it only needs one evaluation to predict the effects of a skill execution, which reduces planning time as well as covariate shift by reducing the number of sequential predictions [75, 115, 156, 204]. Second, a long-horizon skill-level model can leverage a skill's ability to act as a funnel in state space during execution, which simplifies the learning problem.

Collecting diverse and relevant data for training SEMs. To learn accurate and generalizable SEMs, they must be trained on a set of skill execution data that is both diverse and relevant to task planning. While we assume knowledge of the initial state distribution of all tasks, we do not know the distribution of all states visited during planning and execution. As we cannot manually specify this incurred state distribution, we obtain it and train the SEMs in an iterative fashion that interleaves SEM training with data generation by planning and execution, as seen in Figure 4.1. First, given an initial set of skills, we generate single skill execution transitions from the known initial state distribution. This data is used to train the initial SEMs. Then, given a set of training tasks, we use the planner to plan for these tasks using the learned SEMs across a set of initial states. The planner terminates when it finds a path to the goal or reaches a fixed planning budget (reaching maximum number of nodes expanded, maximum search depth, or maximum planning time). Then we sample paths in the graph and simulate them to collect skill execution data, which is added to a dataset of all skill data collected so far. Path sampling is biased toward longer paths and ones that have the newly added skills. The transitions added are filtered for duplicates, since multiple paths in a planning graph may share the same initial segments which would bias the dataset towards transitions closer to the initial states. After a fixed amount of path data is collected, we continue training the SEMs on the updated dataset before restarting the data collection process. In the beginning, it is expected that the planner performance will be highly suboptimal due to the inaccurate initial SEMs. While we use simulation data due to benefits in speed, this is not a requirement and SEMs can be trained with real-world data.

Planning with new skills. The above procedure supports incrementally expanding the list of skills used by the planner. Given a new skill, we first train an initial SEM by sampling from the initial state distribution, then during planning data generation the search-based planner can use the new SEM to get successors. SEMs for new and existing skills will be improved and continuously trained on this new planning data. Fine-tuning previous SEMs is needed, because the new skill might have incurred states that were previously absent from the dataset. Although this fine-tuning may not be necessary in specific cases, we leave detecting such scenarios and reducing overall training budget to future work. Learning one SEM for each skill allows for different parameter spaces (e.g. dimensions, discrete, continuous, mixed) that cannot be easily represented with a shared, common model.

Planning with new tasks. Because the planner does not rely on predefined plan skeletons, it can directly use SEMs to plan for new tasks. Two main factors about data collection affect the generalization capability of the SEMs when applied to unseen test tasks. The first is whether the states incurred while planning for training tasks are sufficiently diverse and relevant to cover the states incurred by planning for test tasks. The second is the planner itself — how greedy is its search and how much it explores the state space. Many planners have hyperparameters that can directly balance this exploration-exploitation trade-off.

4.3.3 Search-based Task Planning

We pose task planning as a graph search problem over a directed graph, where each node is a state x, and each directed edge from x to x' is a tuple (o, θ) such that $f_o(x, \theta) = x'$. Edges also contain the costs of skill executions c_o. During search, this graph is constructed implicitly. Given a node to expand, we iterate over all skills, generate up to B_o parameters per skill that satisfy the preconditions, then evaluate the skill-level dynamics on all state-parameter tuples to generate successor states. B_o decides the maximum branching factor on the graph. This number varies per skill, because some skills have a broader range of potential parameters than others, requiring more samples.

To search on this graph, we apply Weighted A* (WA*), which guarantees completeness on the given graph. If the heuristic is admissible, WA* also guarantees the solution found is no worse than ϵc^*, where c^* is the cost of the optimal path and ϵ determines how greedily the search follows the heuristic. We assume an admissible heuristic is given. This is in line with previous works that have shaped rewards or costs that guide the planner [125, 134, 161, 254].

The proposed method enables planning with new skills and to solve new tasks in continuous states. Planning for new tasks is done by replacing the heuristic and goal conditions, which does not affect the graph construction procedure or the SEMs. Searching in continuous states is more flexible than searching in symbolic states, and it is not necessarily slower. Flexibility comes from the ability to integrate new skills and tasks without needing to create new symbols. Planning speed depends on the size of the action space (branching factor) and the state space. Using symbolic instead of continuous states does not reduce the branching factor, and partitioning continuous states into symbolic states without subgoal skills yield little benefits [104].

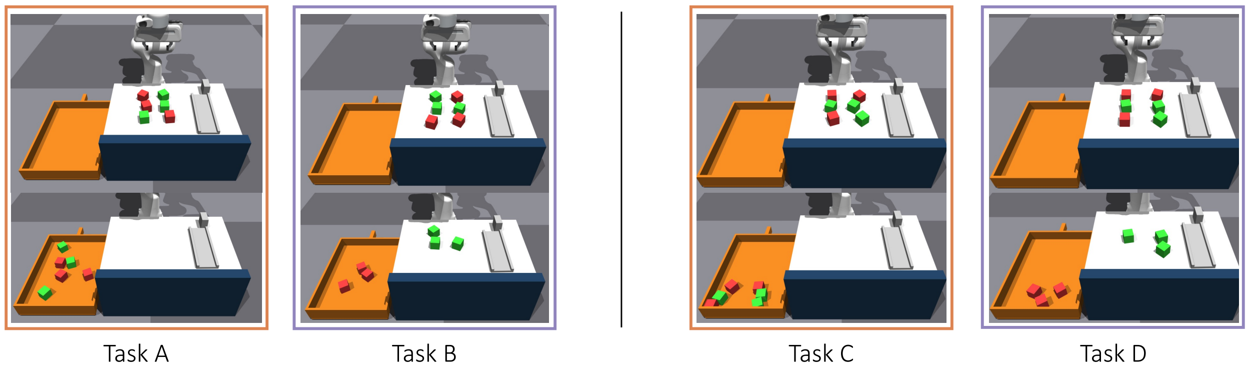


Figure 4.2: Different tasks used in our experiments. The top row shows examples of initial states, the bottom shows examples of goal states. *Left:* blocks to bin tasks (tasks *(A,B)*). *Right:* blocks to far bin tasks (tasks *(C,D))*.

4.4 Experiments

Our main experiment analyzes the effect of incrementally adding new skills to the proposed method on planning performance of both train and test tasks. We apply our method to a block manipulation domain (Figure 4.2) because it can be reliably simulated, contains a diverse set of skills, and the skills have broader applications in desktop manipulation and tool use. In addition, we show our approach compares favorably against planning with simulation and the benefits of using planning data to train SEMs.

4.4.1 Task Domain

The task domain has a Franka Emika Panda 7 DoF arm, a set of colored blocks, a table, a tray, and a bin. On the table, blocks of the same size and different colors are initialized in random order on a grid with noisy pose perturbations. The tray on the table can be used as a tool to carry and sweep the blocks. Beside the table is a bin, which is divided into two regions, the half which is closer to the robot, and the half that is farther away. The state space contains the 3D position of each block, color, and index. We implement the task domain in Nvidia Isaac Gym [164], a GPU-accelerated robotics simulator [129] that enables fast parallel data collection.

Skills. We experiment with four skills: *Pick and Place* (Figure 4.1 skill 1) moves a chosen block to a target location. It has a mixed discrete and continuous parameter space — which object to pick and its placement location. *Tray Slide* (Figure 4.1 skill 2) grasps the tray, moves it to the bin, and tilts it down, emptying any blocks on it into the bin. Its parameter is a continuous value defining where along the length of the bin to rotate the tray. *Tray Sweep* (Figure 4.1 skill 3) uses the tray to perform a sweeping motion along the table. Its parameter specifies where to start the sweeping motion, and the sweep motion ends at the table's edge. *Bin Tilt* (Figure 4.1 skill 4) grasps the handle at the side of the bin and tilts the bin by lifting the handle, which moves blocks in the bin from the close half to the far half. Skills are implemented by following open-loop trajectories defined by the skill parameters. We did not learn more complex skills as our work focuses on task planning and not skill learning.

Tasks. We evaluate on four different tasks (Figure 4.2) that are variations of moving specific sets of blocks to different regions in the bin. Two tasks are used to collect SEM training data: *Move All Blocks to Bin (A)* and *Move All Blocks to Far Bin (C)*, while the remaining two are used to evaluate learned SEMs: *Move Red Blocks to Bin (B)* and *Move Red Blocks to Far Bin (D)*. Each task uses the same background cost function, which is the distance the robot's end-effector travels, plus a small penalty for placing the gripper inside the bin. The admissible heuristic used is the mean distance of each block to the closest point in their target regions.

While *Pick and Place* can make substantial progress on all tasks, it alone is not sufficient because kinematic constraints inhibit the robot from directly placing blocks on the far side of the bin, so *Bin Tilt* or *Tray Slide* is needed. Additionally, using other skills can achieve lower costs; *Tray Sweep* can quickly move multiple blocks into the bin, but this may move blocks that need to stay on the table. The sequence of skills may change depending on the initial placement of the blocks, and the path needs to be low-cost.

4.4.2 Lifelong Task Planning Results

To evaluate our approach for lifelong integration of new skills, we add the four skills over time using the iterative training procedure. We evaluate two scenarios, first in which the train-test task pair are respectively tasks A and B, and second with C and D. In each case, the robot starts with only *Pick and Place*, while *Tray Slide*, *Tray Sweep*, and *Bin Tilt* are added successively in that order at pre-determined intervals. We measure planning performance using execution costs, execution success, and planning time. For each goal, the robot plans only once from the initial state, which terminates when it succeeds or times out.

Figure 4.3 plots the execution costs over time for both scenarios. The proposed method is able to incorporate new skills over time, lowering execution costs when applicable by planning with new skills. For example, adding *Tray Slide* allows the planner to find plans with significantly reduced costs across all tasks, since multiple blocks can now be moved together. In other cases, adding a new skill does not affect task performance. One example is adding *Bin Tilt* to the blocks to anywhere in bin tasks *(A,B)*, because the main use of the skill is to move blocks to the far side of the bin. Another is on adding *Tray Sweep* — it significantly reduced costs for moving all blocks to the bin *(A,C)*, but less so for moving only red blocks to the bin *(B,D)*. This is because sweeping is only useful for the latter task when multiple red blocks line up in a column near the bin, which rarely occurs in the randomly initialized states.

Figure 4.4 plots the success rate of finding successful plans (dashed) and optimal plans (solid) with new skills. Immediately after adding a new skill, there is insufficient data to learn a robust SEM, so the planner is unlikely to find optimal plans using the new skill. Or, if it does find a plan, the plan often leads to execution failures. As more data is collected, SEM accuracy improves and the probability of finding optimal plans increases. Figure 4.4 also shows how some tasks can only be completed after a new skill is incorporated. For instance, with just *Pick and Place*, the robot can accomplish blocks to bin tasks *(A,B)*, but fails to plan for the blocks in far bin tasks *(C,D)*. Adding new skills for *(A,B)* did not change the success rate of the task, which remained at 100%, although the composition of the plans found does change. For *(C,D)*, adding *Tray Slide* enabled 100% success rate, while adding *Tray Sweep* did not affect plan compositions, but adding *Bin Tilt* did. These results show that our proposed method can learn skill effects and plan with SEMs in a lifelong manner, and that SEMs can plan for new tasks without additional task-specific learning. Qualitative results can be found in Appendix 9.3.5.

Task	Sim	SEMs (Ours)
A	776.19 (46.9)	1.3 (0.7)
C	1736.8 (187.)	0.98 (0.3)

Table 4.1: Comparing plan times in seconds using simulator vs. SEMs. Parenthesis indicate standard deviations.

Task	Pick-Place	+Tray-Slide	+Tray Sweep	+Tilt Bin
A	11.3 (3.4)	20.2 (7.9)	0.6 (0.5)	1.3 (0.7)
B	7.4 (2.3)	14.9 (8.2)	18.0 (14.3)	22.1 (12.4)

Table 4.2: Plan times (seconds) using SEMs for objects to bin tasks (A, B) with an increasing number of skills.

	Success	Cost
Pick and Place	1.0	6.68 (0.3)
+Tray Slide	0.9	3.9 (0.9)
+Tray Sweep	0.8	2.61 (0.7)

Table 4.3: Real-world results on *Red Blocks to Bin*. Costs: mean (std).

Planning with a Simulator. To highlight the need for learning SEMs instead of simulating skill effects for task planning, we compare their planning times in Table 4.1. We only benchmarked cases where the tasks are about moving all blocks and all skills are available. On average, using the learned model takes less than a second while using the simulator takes ten minutes to half an hour. Note that these results leverage the simulator's ability to simulate many skill executions concurrently. Thus, using the simulator for more complex scenarios is prohibitively time consuming due to 1) the large branching factor and 2) a skill's extended horizon, which is much longer than single-step low-level actions or short-horizon motion primitives. Additionally, Table 4.2 shows the plan times for SEMs with increasing number of skills. In all cases our planner find plans in less than half a minute.

Training on Planning Data vs. Random Data. To evaluate the benefits of using planning data for the iterative training of SEMs, we compare the test-task success rate between our approach and one that generates data by executing random skill sequences. See results in Figure 4.5. Training on planning data achieves higher success rates using fewer samples than training on random data does, illustrating the benefit of guiding data collection using a planner.

Real-world Results. We built our task domain in the real world (test tasks in Figure 4.1) and used the learned SEMs to plan for the test task *B*. Three sets of planning experiments were performed, one with only `Pick and Place`, one with the addition of *Tray Slide*, and one with the addition of *Tray Sweep*. Each set of experiments in Table 4.3 consists of 10 planning trials with different initial block configurations. These results are similar to the ones shown in the task *A* test curves in Figure 4.3. The differences are due to the small changes in real-world object locations and controller implementations. While we did not fine-tune SEMs on real-world data, doing so may improve real-world performance.

4.5 Conclusion

We propose using search-based task planning with learned skill effect models (SEMs) for lifelong robotic manipulation. Our approach relaxes prior works' assumptions on skill and task representations, enabling planning with more diverse skills and solving new tasks over time. Using SEMs improves planning speed, while the proposed iterative training scheme efficiently collects relevant data for training.

In future work, we will scale our method to larger number of skills and parameters by using partial expansions and learned parameter samplers. We will also explore estimating model uncertainty, using that to both steer planning away from uncertain regions and also fine-tune existing SEMs only on data about which the models are sufficiently uncertain.

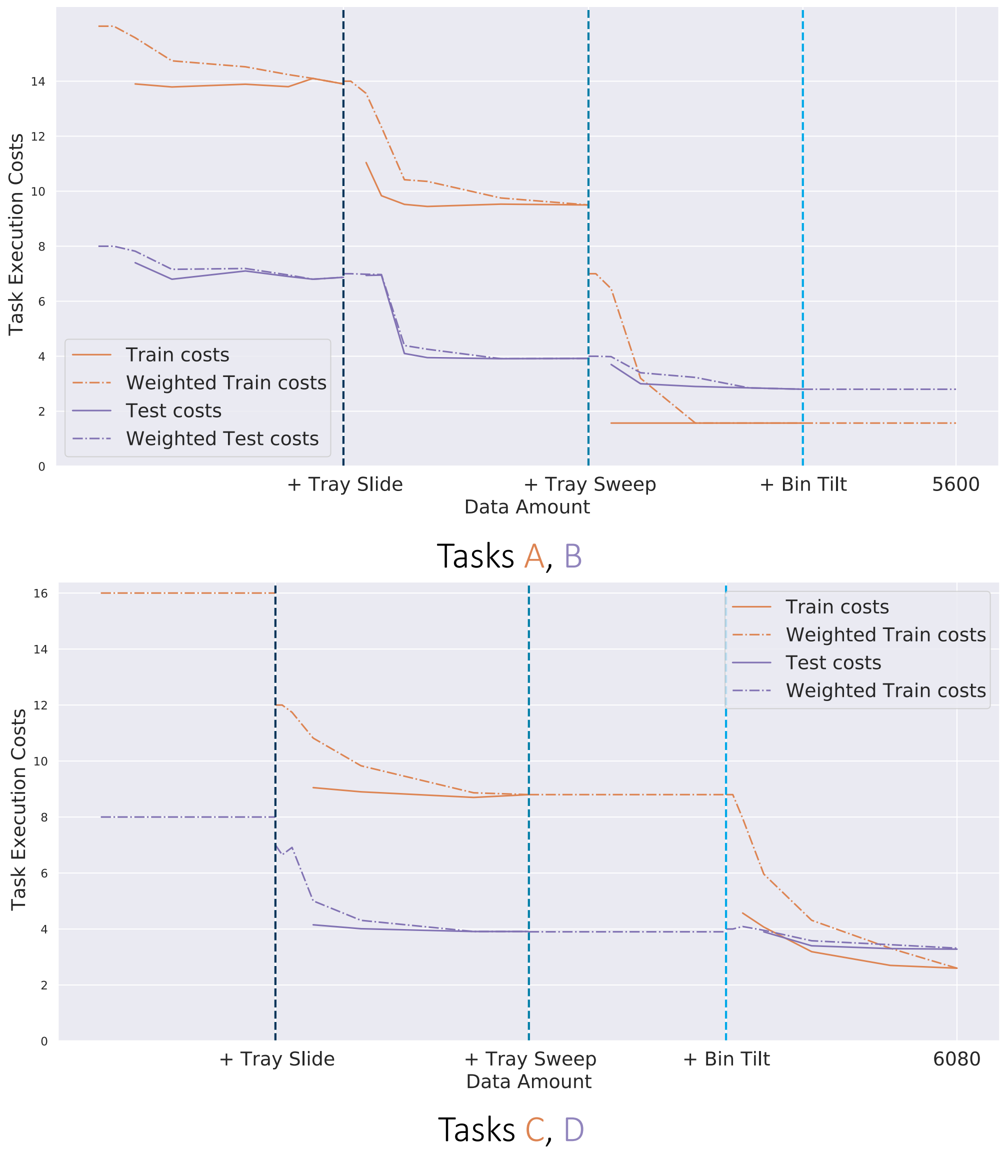


Figure 4.3: Task execution costs plotted over time as new skills are learned and integrated in a lifelong manner. Blue vertical lines signify the addition of a new skill. Weighted costs are calculated by weighting the task cost with the success rate.

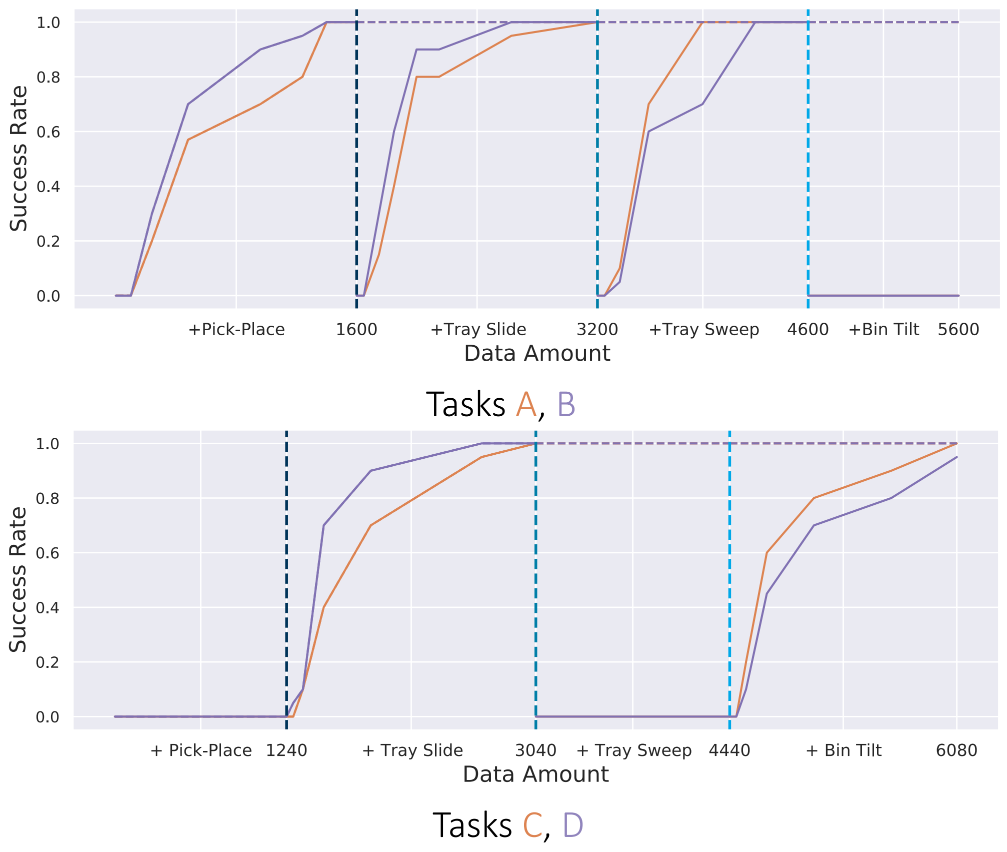


Figure 4.4: Task execution success rate for each new added skill. Each skill is being added over time. Orange are train tasks; purple are test tasks. Solid lines are planning with new skills; dashed are with any skills.

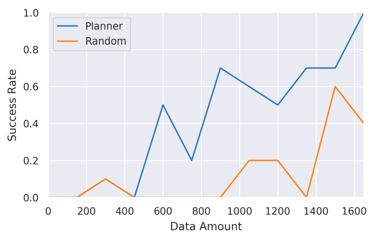


Figure 4.5: Success on task *B* with SEM trained on random vs. planner data.

Part III

Reusing Visual Representations

In this next set of works we develop techniques that reuse visual representations for learning different robot manipulation tasks. Similar to previous works, our main aim with these works is to efficiently learn robust robot manipulations. For this, Chapter 5 develops a common visual representation for learning manipulation skill preconditions. We decompose manipulation tasks into objects and their interactions. Large-scale object interactions generated in a simulator is used to learn a common 3D representation. This common representation is then used to efficiently learn robot skill preconditions from very little real-world data. For our next set of works we focus on reusing web-trained visual representations for robot manipulation. Chapter 6 proposes *RoboAdapters* for efficient adaptation of pretrained visual representations for robot manipulation. RoboAdapters maintain the original capabilities of pretrained vision model and allow them to be adapted for a large number of tasks over time. Chapter 7 proposes a multi-resolution sensing architecture to enable real-time with lage pretrained vision-language models.

5 Relational Learning for Skill Preconditions

This chapter is based on [212].

Abstract: *To determine if a skill can be executed in any given environment, a robot needs to learn the preconditions for the skill. As robots begin to operate in dynamic and unstructured environments, precondition models will need to generalize to variable number of objects with different shapes and sizes. In this work, we focus on learning precondition models for manipulation skills in unconstrained environments. Our work is motivated by the intuition that many complex manipulation tasks, with multiple objects, can be simplified by focusing on less complex pairwise object relations. We propose an object-relation model that learns continuous representations for these pairwise object relations. Our object-relation model is trained completely in simulation, and once learned, is used by a separate precondition model to predict skill preconditions for real world tasks. We evaluate our precondition model on 3 different manipulation tasks: sweeping, cutting, and unstacking. We show that our approach leads to significant improvements in predicting preconditions for all 3 tasks, across objects of different shapes and sizes*

5.1 Introduction

Skill *preconditions* are necessary for a robot to know when a given skill can be executed across different situations. For robot manipulators to operate in complex and unstructured environments, precondition models need to adapt to variable number and type of objects in the scene, *e.g.*, preconditions for food cutting should generalize between apples and carrots. Additionally, since collecting large amounts of real world data for every skill is impractical, we also require precondition models to learn from few samples. To achieve the above requirements, we observe that many complex manipulation tasks often require specific relations between a number of objects to be valid. Thus, these tasks can often be simplified by decomposing them into less complex object interactions. We focus on modeling pairwise relations between objects. Moreover, these object relations are also often shared across multiple tasks. Thus, we also aim to learn a common representation for these relations, which can be directly used by our precondition models.

Our work uses a compositional approach for precondition learning. We learn precondition models from a few different example scenes. Each scene is decomposed into its constituent objects and their relations as shown in Figure 5.1. Learning precondition models from such a structured representation requires us to infer object identities and pairwise object relations. While object labels can be found using state-of-the-art algorithms [62], we focus on learning representations for object pair relations. Most of the prior work uses discrete representations for object relations [47, 190, 261]. These discrete relations are mostly fixed

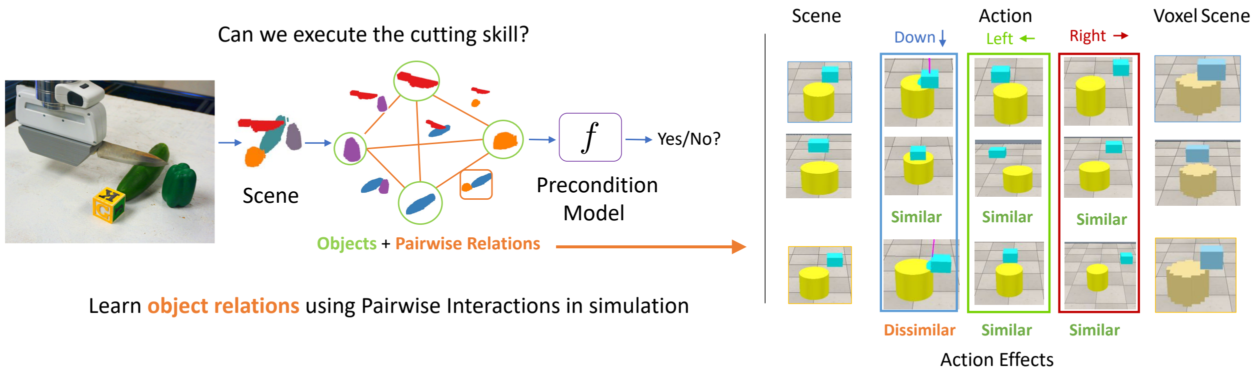


Figure 5.1: Overview of the overall precondition learning process. We learn precondition model by decomposing a scene into its objects and relations. To learn continuous representations for object relations we utilize simulations to observe how objects interact with each other.

and often insufficient to distinguish between similar scenes. For instance, in Figure 5.1, even though both relations might be classified as "above" the semantics of each scene are very different. Although recent approaches have looked at learning continuous representations for object relations [85, 147], these approaches use a pre-defined set of relations. They further assume access to binary labels which allows them to cluster scenes with similar relations together.

Instead of assuming a fixed set of relations we propose a novel data-driven approach to learn continuous representations for object relations. We leverage simulations to allow for unconstrained interactions between object pairs. Using simulations provides us access to a large set of labeled data underlying these interactions, such as object contacts, pose changes. We use these interaction effects to create a contrastive loss formulation that allows us to learn continuous representations for object relations. Our main contributions include 1) We propose a novel approach that leverages simulations to learn a continuous representation for object relations. 2) Our approach is grounded in object interactions and hence does not assume a fixed set of object relations. 3) We show that our continuous representations, which are learned via simulation, can be directly used for precondition learning of real world manipulation tasks, without requiring any specific sim2real adaptation.

5.2 Related Works

Learning skill preconditions is closely related with symbol learning. Symbolic language formulations explicitly include pre-conditions and post-conditions [145]. Thus previous approaches have looked at inferring discrete symbols and predicates from continuous states. There have been two sets of approaches in this regard. At one end, previous works have looked into classifying continuous states into pre-defined symbolic forms [15, 47, 89, 174]. Alternately, other works aim to discover these discrete forms by methods such as clustering [77, 114, 172, 190]. Some of the above approaches have also looked into learning discrete representations for object relations. For instance, [47] uses histogram features extracted from point clouds to classify object relations into discrete categories. In [190], the authors construct a graph of contact points between two objects, which is used to classify object relations using supervised learning. While [261] learns discrete spatial relations by geometrically dividing the space around an object into a fixed set of regions. In contrast to discrete symbolic forms, we focus on learning continuous representations for object relations. Also, instead of solely focusing on either geometric relations [261] or contact points [190], our approach is grounded in a robot's interaction with pairs of objects. This allows us to learn continuous relational representations by leveraging spatial geometry, contact information, and object-interaction dynamics all together.

Recent work has also focused on learning continuous representations for object relations [85, 147]. Both [85, 147] learn these representations from a dataset of $N \sim 500$ scenes. Each scene has a pair of objects, which are sampled from a set of known object templates. The object pair arrangement is sampled from a pre-defined set of relations and the aim is to generalize these arrangements to new sets of objects. The dataset is manually labelled with binary similarity value for each scene pair [147]. Based on this dataset [147] proposes a distance metric to extract similar scenes. While [85] uses contrastive learning to learn this distance metric. Although [85, 147] learn continuous relationas, the dataset used to learn these consists of a pre-defined set of discrete relations only. Additionally, using only binary labels to distinguish between scenes is limiting, *e.g.* scenes where object A is on the left of object B are labeled similarly irrespective of the distance between two objects. By contrast, rather than manually labelling scenes, we leverage simulations to generate large amount of object pair interactions. We group similar scenes together based on effets of these interaction. Since these effects have continuous values, our model is able to learn richer relational representations. Further, since these interactions are randomly generated, we do not assume any predefined set of relations.

Previous work [111] has also looked into precondition learning using objects, parts and their interactions. In [111] these are referred to as scene elements and represented by their mean 3D position. All scene elements are then used together for spatial precondition learning using random forests. However, using position features alone is severely limiting since many manipulations tasks depend on object sizes, orientation, geometry, and specific contact distributions.

5.3 Approach

We learn skill preconditions by decomposing a scene S into its constituent objects $(o_1, o_2, o_3, \ldots)$ and their continuous relations $(r(o_1, o_2), r(o_1, o_3), \ldots)$. We assume the scene decomposition is already known and object identities can be determined if required. To infer object relations $r(o_i, o_j)$, we learn a function $f_{\text{rel}} : \mathbb{R}^{C \times L \times W \times H} \rightarrow \mathbb{R}^K$, which uses a 3D voxel based perceptual input, where C is the number of channels and L, W, H are the length, width and height of the scene, and outputs a continuous relational embedding of size K. This perceptual input consists of a pair of objects in the scene. To learn f_{rel}, we leverage simulations to generate a large set of pairwise object interactions and use their observed effects as our learning signal. More specifically, we assume the existence of a set of simple perturbation actions that allow a robot to move an object around. These actions let the robot freely interact with pairs of objects. The use of simulation also gives us access to the underlying effects of these interactions *e.g.*, change in pose, object contacts and normals, and force-torque values. We utilize these observed effects to create a contrastive learning formulation which groups scenes together based on the similarity of these interaction effects.

Once we learn f_{rel}, we use it to extract object relations $r(o_i, o_j) = f_{\text{rel}}(o_i, o_j)$ for all object pairs in the scene. These extracted relations are then used as input to our precondition learning model, which is trained directly on real world manipulation task data. While training the precondition model, we do not fine-tune the object relation model f_{rel}. We show that f_{rel} trained in simulation can be directly transferred to the real world without requiring any sim2real adaptation.

5.3.1 Generating Pairwise Interactions In Simulation

Pairwise object relations are closely intertwined with their potential interactions. For instance, in Figure 5.1 (right) if an object is gently placed on another object it will either stay in place or it might tilt over and fall. Although both of these scenes might classify a discrete object relation as being "above", the exact semantics of these relations are quite different. To generate object pair interactions we use the V-REP simulator [188] with the underlying Bullet physics engine (v2.83) [33]. For each interaction, we initially create a scene with two objects o_i and o_j. Each object's shape is chosen from a fixed set of primitive shapes including cuboids, cylinders and spheres. We generate a voxel representation for this scene using an object-centric approach, *i.e.*, the reference frame of the scene is centered on one of the objects, referred to as the anchor object (o_i), while o_j is the the referrant object. Figure 5.1 (right column) shows an example of such a voxel based representation. To observe the interaction effects of these two objects we keep the anchor object static.

Since objects in the real world occur in varying spatial locations we do not assume any fixed object positions. However, since the scene reference frame is centered on the anchor object we do not need to set its position and orientation. To position the referrant object we sample a location (x, y, z) around the anchor object such that the referrant object is less than 0.5 meter away from the anchor. To set the orientation we rotate the object around the z-axis between $(-\pi/6, \pi/6)$ radians.

To create object pair interactions, we apply local perturbations to the referrant object. These local perturbations move the referrant object in the Euclidean space. Figure 5.1 (right column) shows three different instances of local perturbations. Formally, these perturbation can be expressed in polar format as (r, θ, ϕ), where r is the action magnitude and θ, ϕ, represents the action direction. We sample action directions that are axes aligned with the objects reference frame as well as along the diagonals of each axis pair. We use two different strategies to sample action magnitude. First, we use a fixed action magnitude sampled from $[5cm, 20cm]$. Additionally, we also use adaptive actions, wherein the action magnitude is set as the distance between the anchor and the referrant object centers. These adaptive actions ensure that objects will interact even if the distance between them is greater than the maximum fixed action magnitude. To perform local perturbation actions, we add a virtual robot with a spring-damper system.

Given the data generation process, we generate $N \sim 100,000$ pairwise object scenes. For each scene, we save the voxel representation for the anchor and referrant objects. Additionally, for each action we also record the distance moved (in meters) by the referrant object (Δp), as well as the change in orientation (in radians) of the referrant object $(\Delta\theta)$ when it interacts with the anchor object. For scenes which involve contacts between objects we also record the contact position and the contact normals. These contact positions are recorded for the anchor object relative to its frame of reference. More specific details can be found in the supplementary material.

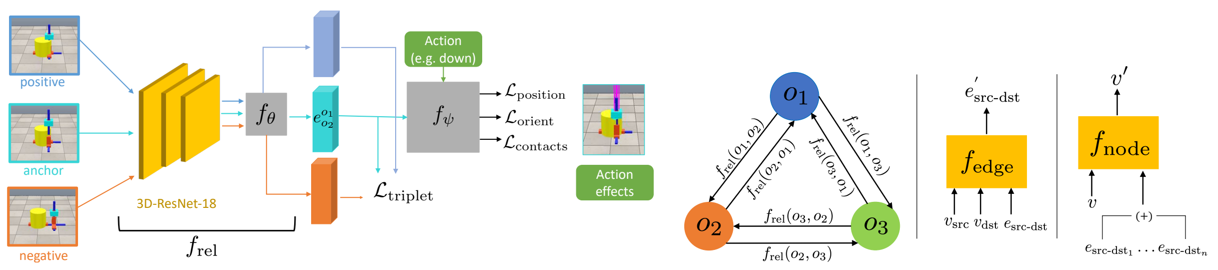

positive
anchor
negative
3D-ResNet-18
f_{rel}
f_θ
$e^{o_1}_{o_2}$
Action (e.g. down)
f_ψ
$\mathcal{L}_{\text{position}}$
$\mathcal{L}_{\text{orient}}$
$\mathcal{L}_{\text{contacts}}$
$\mathcal{L}_{\text{triplet}}$
Action effects
o_1
o_2
o_3
$f_{\text{rel}}(o_1, o_2)$
$f_{\text{rel}}(o_2, o_1)$
$f_{\text{rel}}(o_3, o_1)$
$f_{\text{rel}}(o_1, o_3)$
$f_{\text{rel}}(o_3, o_2)$
$f_{\text{rel}}(o_2, o_3)$
$e'_{\text{src-dst}}$
f_{edge}
v_{src} v_{dst} $e_{\text{src-dst}}$
v'
f_{node}
v
(+)
$e_{\text{src-dst}_1} \cdots e_{\text{src-dst}_n}$

Figure 5.2: *Left:* Overall architecture to learn object relation model f_{rel}. *Right:* Precondition learning architecture based on GNNs, shows graph model at initialization and the graph edge and node models.

5.3.2 Learning Object Relations

We use pairwise object interactions to learn our object relation model f_{rel}. The input to f_{rel} is 3D voxel representation of an object pair and it outputs a continuous low dimensional relation embedding. This formulation does not require an action as an input and hence f_{rel} can be used directly at test time. Our main insight to learn f_{rel} is based on the fact that the same actions on similar scenes should result in similar effects. These action effects refer to change in properties such as object position, orientation, contacts. We use these action effects to train f_{rel} using two different sets of losses. First, we take a metric learning approach and use the action effects to create contrastive losses that group similar scenes together and away from dissimilar ones. Second, we combine the output relation embedding with the action representation to directly predict the action effects.

Model: Figure 5.2 (Left) shows the architecture of our model. We use a ResNet [63] based architecture as our backbone model to process the 3D voxel input. We use [59] to adapt the ResNet model to operate on 3D input. The output of the model is projected down to $\mathbb{R}^{K=256}$ using a linear projection. We refer to the output of the linear projection as the relation embedding $r(o_i, o_j) = f_{\text{rel}}(X)$, where o_i, o_j denotes the anchor and referrant object respectively and X is the perceptual input with the object pair. The projected relation embedding $r(o_i, o_j)$ is then concatenated with the action input a. We create a by combing the action vector of size $\mathbb{R}^3$ with a one-hot label of size $\mathbb{R}^2$, this label is used to differentiate between fixed and adaptive perturbation actions. This combined representation is then used to predict a set of outputs which correspond to the observed effects of this action.

Loss: We learn f_{rel} using contrastive losses, which have been used to learn continuous embeddings from high dimensional data [58, 85, 165, 200]. Contrastive losses are useful for our formulation since they compare sample similarity directly in the representation space. Amongst the multiple contrastive loss formulations we use the triplet loss [200]. Formally, the triplet loss requires an anchor (a) and positive (p) - negative (n) pairs. The aim is to keep the anchor and positive samples together while pushing the negative sample away based on a margin, $L_{\text{triplet}} = (d_{ap} - d_{an} + \gamma)_+$ where d_{ap} and d_{an} are the squared euclidean distance between the anchor and positive embedding, and anchor and negative embedding respectively. γ is a constant margin between similar and dissimilar pairs.

We utilize the observed effects of the perturbation actions to get positive and negative samples for the anchor scene. Specifically, scenes where all local perturbations lead to similar effects are more similar than other scenes. Based on this we add triplet loss for both position (Δp) and orientation changes $(\Delta\theta)$ separately. To get positive samples for $(\Delta\theta)$, we compare all action effects using $|\Delta\theta^i_1 - \Delta\theta^i_2| < \Delta\theta_{\text{sim}}$,

where $\Delta\theta_1^i$, and $\Delta\theta_2^i$ are the effects of i'th local perturbation action on two different scenes. To achieve the same for position changes, instead of using Δp directly we use the ratio of observed change in position to the desired change in position $\Delta p^r = \frac{\Delta p^{\text{observed}}}{\Delta p^{\text{desired}}}$, where $\Delta p^{\text{observed}}$ is the observed change in referrant object center position, and $\Delta p^{\text{desired}}$ is the desired change *i.e.* action vector. We set $\Delta p^{\text{desired}}$ to 1 for normal actions, while for adaptive actions it is set to the voxel distance between objects centers. This is required to allow adaptive actions to be compared against each other since these actions result in different $\Delta p^{\text{desired}}$ values across different scenes. We sample negative scenes by using $|\Delta\theta_1^i - \Delta\theta_2^i| > \Delta\theta_{\text{diff}}$, where we set $\Delta\theta_{\text{diff}} = \Delta\theta_{\text{sim}} + \epsilon$, where ϵ is a small value set manually. We include implementation details, including the different hyper-parameters, for the contrastive loss, in the supplementary material. In addition to contrastive losses, we also use the following prediction losses to train f_{rel}:
Predicting object positions: We use f_{rel} to predict the change in object position Δp for the referrant object. However, since we use voxels for our scene representation we lose the granularity of continuous movement in the euclidean space. Hence, instead of predicting the raw values of change in position we predict the voxel displacement of the referrant object's origin Δp^v. We add a mean-squared loss on this prediction, $L_{\text{pos}} = ||\Delta p^v_{\text{pred}} - \Delta p^v_{\text{gt}}||_2$

Predicting object orientations: We also predict the change in orientation $\Delta\theta$ of the referrant object. Since the object is attached to a robot and moved controllably the magnitude of $\Delta\theta$ is usually low $\Delta\theta \sim 0.1$. Hence, instead of using L_2 loss we use L_1 loss, $L_{\text{orient}} = ||\Delta\theta_{\text{pred}} - \Delta\theta_{\text{gt}}||_1$

Predicting contact distributions: We also predict the contact distributions between interacting objects. Given the contact data from simulation we fit a Gaussian model to predict the mean (x, y, z) of the contact point distribution. We then add the following loss, $L_{\text{contact}} = ||(\mu_{\text{pred}} - \mu_{\text{gt}}||_2$.

5.3.3 Learning Precondition Models

Once we learn the object relation model f_{rel} from pairwise object interaction data we use it for downstream real-world precondition learning tasks. Our precondition learning model is based on deep neural network architectures which can operate on a set of inputs which are provided by the learned f_{rel} model. Formally, given a scene S with objects $S = \{o_1, o_2, \ldots, o_N\}$. We use f_{rel} to get object relation embeddings for all $N \times (N-1)$ object pairs $f_{\text{rel}}(o_i, o_j)$, where o_i and o_j are the anchor and referrant object. For our precondition learning model we use two different network architectures *i.e.* Relational Networks (RNs) and Graph Neural Networks (GNNs). RNs have been used to reason about objects and their relations [194]. GNNs are more general neural network architectures where the input can be represented as a graph $G = (V, E)$, with vertices V and edges E [13, 102, 197]. Figure 5.2 (Right) shows our formulation of a scene S as a GNN. Specifically, we represent each object as a separate node $v_i \equiv o_i$ and add bi-directional edges between every node pair. At initialization, we add $f_{\text{rel}}(o_i, o_j)$ as the edge information for edge $e_{ij} = (v_i, v_j)$ Unless explicitly specified, we do not add any node information to the vertices. To allow for better inference, we stack two layers of our graph network architecture. The output of the final graph layer is classified using the sum of all the node and edge embeddings. More details are presented in the supplementary material.

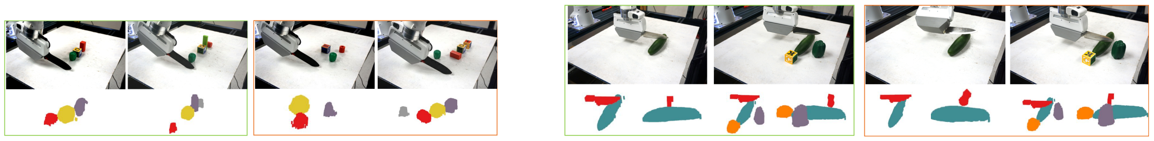


Figure 5.3: *Left:* Example scenes with 3 and 4 objects used for the objects in line task. *Right:* Example scenes with 0 and 2 distractor objects for learning food cutting preconditions.

5.4 Experiments

Our aim is to investigate the following questions, 1) Given limited amount of precondition training data, how effective are the embeddings learned by our object relation model for precondition learning? 2) Does a structured representation of a scene as objects and relations help in precondition learning for scenes with variable number of objects? and 3) How effectively do the precondition models generalize to objects with different shapes and sizes than the train set? To verify these we use three different manipulation tasks: sweeping objects in a line, food cutting, and 3D block unstacking. Moreover, to show the effectiveness of our learned embeddings, we do not finetune the object relation model f_{rel} during precondition learning.

Metrics: Since precondition learning is a binary prediction problem we compare models using F1 and weighted F1 scores. F1 score is the harmonic mean of precision and recall with higher values (max: 1.0) indicating better models and wt. F1-score is used to account for data imbalance.

Baselines: We compare our approach against both learning and non-learning baseline methods. Among learning based methods we initially use models that use 3D scene input and output the binary precondition. We use multiple baseline architectures to represent this learned model. First, similar to our object relation model, we use a 3D ResNet-18 model with the output being projected down using multiple linear layers to predict the precondition output. To increase the representation capacity of this baseline model we also evaluate a ResNet-34 based model which is adapted in the same way as [59]. Given the limited amount of real world data, we also compare against a smaller VGG [223] based model with 3D convolutions.

To verify the effectiveness of continuous relations we also use a discrete relation baseline, wherein instead of continuous relations we use discrete relations. We refer to this as *DiscreteRel*. Finally, in addition to learning based methods we also evaluate another approach which utilizes the simulator to verify the preconditions. In this approach we transfer the 3D representation of a real world scene into the simulator. Once the scene has been transferred we simulate the task to verify the preconditions. We refer to this baseline as *real2sim*. See appendix for training and implementation details.

5.4.1 Sweeping Objects in Line

Humans are adept at inferring if objects lie in an approximately straight line. However for robots, the presence of sensor noise, variances in object sizes and small amount of input data make this a challenging task. Specifically, we look at task preconditions when a robot arm sweeps along the X-axis. Figure 5.3 (left) shows some examples from the train dataset. We collect scenes with variable number of objects (3, 4 and 6) and train on 25 different scenes. Table 5.1 (left) shows results for multiple train-test sets. For the first train-test split all methods perform well. However, for the other splits *i.e.*, when we test on 6 objects and objects with different shapes and sizes, the learning baseline methods perform poorly (wt. F1-score

	Sweep Objects In Line				Food Cutting			
Model	Train Set (objects)	Test Set (objects)	F1	Wt-F1	Train Set (distractors)	Test Set (distractors)	F1	Wt.-F1
ResNet-18	3, 4	4	0.934	0.951	0, 1	2	0.667	0.741
Resnet-34	3, 4	4	0.934	0.951	0, 1	2	0.701	0.741
VGG*	3, 4	4	0.911	0.948	0, 1	2	0.720	0.590
DiscreteRel	3, 4	4	0.96	0.96	0, 1	2	0.38	0.456
Real2Sim	3, 4	4	0.869	0.871	-	-	N/A	N/A
Our Model (RN)	3, 4	4	0.949	**0.970**	0, 1	2	0.649	0.720
Our Model (GNN)	3, 4	4	0.921	0.944	0, 1	2	0.841	**0.804**
ResNet-18	3, 4	6	0.802	0.771	0,1,2	3	0.844	0.902
ResNet-34	3, 4	6	0.823	0.833	0,1,2	3	0.777	0.871
VGG*	3, 4	6	0.667	0.481	0,1,2	3	0.788	0.842
DiscreteRel	3, 4	6	0.98	**0.98**	0,1,2	3	0.200	0.658
Real2Sim	3, 4	6	0.960	0.960	-	-	N/A	N/A
Our Model (RN)	3, 4	6	0.971	**0.981**	0,1,2	3	0.880	0.935
Our Model (GNN)	3, 4	6	0.948	0.969	0,1,2	3	0.921	**0.940**
ResNet-18	3, 4	6 (diff. size)	0.695	0.662	0,1,2	4	0.827	0.866
ResNet-34	3, 4	6	0.692	0.640	0,1,2	4	0.833	0.868
VGG*	3, 4	6	0.640	0.601	0,1,2	4	0.720	0.770
DiscreteRel	3, 4	6	0.782	0.801	0,1,2	4	0.400	0.590
Real2Sim	3, 4	6	0.904	0.912	0,1,2	4	N/A	N/A
Our Model (RN)	3, 4	6	0.952	**0.952**	0,1,2	4	0.929	0.944
Our Model (GNN)	3, 4	6	0.952	**0.952**	0,1,2	4	0.960	**0.960**

Table 5.1: Precondition learning results for sweeping objects in a line and food food skill.

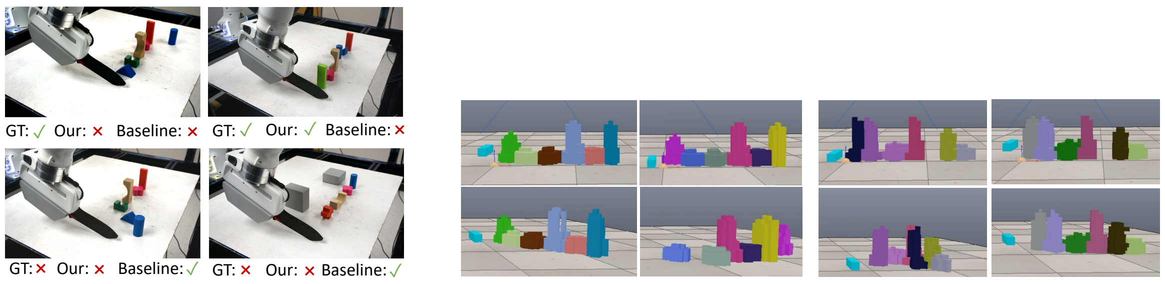


Figure 5.4: *Left:* Test set examples with *different* shapes and sizes for sweeping task. Baseline here referrs to visual baseline. *Right:* Example scenes for sweeping task when imported using the real2sim baseline. The top row shows the initial scene before the sweep while the bottom row shows the after scene.

Model	Train Set	Test Set	F1	Wt-F1	Train Set	Test Set	F1	Wt.-F1
ResNet-18	3,4,5	7	0.741	0.836	3,4,5,7	6	0.770	0.820
ResNet-34	3,4,5	7	0.712	0.809	3,4,5,7	6	0.769	0.820
VGG*	3,4,5	7	0.711	0.804	3,4,5,7	6	0.744	0.818
DiscreteRel	3,4,5	7	0.561	0.655	3,4,5,7	6	0.594	0.617
Real2Sim	3,4,5	7	0.671	0.732	3,4,5,7	6	0.651	0.717
Our Model (RN)	3,4,5	7	0.685	0.775	3,4,5,7	6	0.679	0.771
Our Model (GNN) all edges	3,4,5	7	0.825	0.869	3,4,5,7	6	0.819	0.857
Our Model (GNN) sparse edges	3,4,5	7	0.864	**0.898**	3,4,5,7	6	0.866	**0.894**

Table 5.2: Results for block unstacking task with two different train-test splits.

of 0.83 and 0.66). This is not unexpected given that we have limited training data and a large mismatch between train and test distributions. While, discrete relations baseline performs quite well on the first two splits but poorly on the last split. Also, real2sim is unaffected by the distribution mismatch and performs quite well with a small decrease in its performance on the 3rd split. Both of the latter baselines perform poorly due to more complex geometries of objects in the test set. This leads to imperfect voxel representations, which results in incorrect discrete relations and noisy simulation results when exported into the simulator respectively. We visualise this in Figure 5.4 (Right) which shows two different sets of scenes, each containing two scenes with similar initial configurations but very different final positions. Alternately, both of our models are able to outperform the baseline methods with a wt. F1-score of 0.96 and 0.95 respectively. Note that we only input object relation embeddings to our model and no other object specific information.

5.4.2 Food Cutting

We use cutting food as our next task since it requires reasoning about multiple types of interactions. For instance, to cut a food item the robot needs to hold the knife with its sharp edge in close contact with the food item, the knife should be oriented correctly to pass through the food item, and there should be no obstacle that can potentially hamper the back-and-forth cutting motion. Since the simulator does not implement cutting we *cannot* use the real2sim baseline. Figures 5.3 (right) shows some samples of the collected data. Table 5.1 (right) shows results for this task. For the initial case, wherein we train with only one distractor object and test on more than one distractor objects, all methods perform poorly with a maximum wt. F1-score of our GNN based model 0.804. We believe this happens because with just one distractor in the train set many different object permutations were never observed and hence the model performs poorly on the test set. However, when we train models with upto 2 distractors the performance on test sets with 3 or 4 distractors is much better. Also, for all of the train-test splits our relational precondition models are able to outperform all other baseline methods (wt. F1: 0.960 and 0.940). Among our object relation based precondition models, the GNN based models perform slightly better. This can be attributed to their larger representation capacity, since we stack two GNN layers while the relation network model only contains one relational layer.

Model	F1	Wt-F1
ResNet-18	0.695	0.719
ResNet-34	0.714	0.673
VGG*	0.545	0.665
DiscreteRel	0.200	0.355
Real2Sim	0.827	0.817
Our Model (RN)	0.697	0.736
Our Model (GNN) all	0.923	**0.919**
Our Model (GNN) sparse	0.923	**0.919**

Table 5.3: Results for precondition learning of box stacking task with completely different blocks (objects) in the test set.

Model	Task	F1	Wt-F1
Only predictive loss	Cutting Food	0.72	0.78
Only Δp and $\Delta\theta$ predictive loss		0.72	0.78
Only triplet loss		0.828	**0.866**
Only predictive loss	Block Unstacking	0.802	0.824
Only Δp and $\Delta\theta$ predictive loss	no contacts	0.775	0.800
Only triplet loss		0.835	**0.849**
Using mean position		0.681	0.721
Using mean position + bounding box		0.776	0.816
Only position loss (pred. + cont.)		0.672	0.759

Table 5.4: Ablation results for different losses. The first three rows show values for food cutting task with 2 distractor objects in training and 4 the test set. The next 3 rows show block unstacking results with 3 to 5 objects in train set and 7 in the test set.

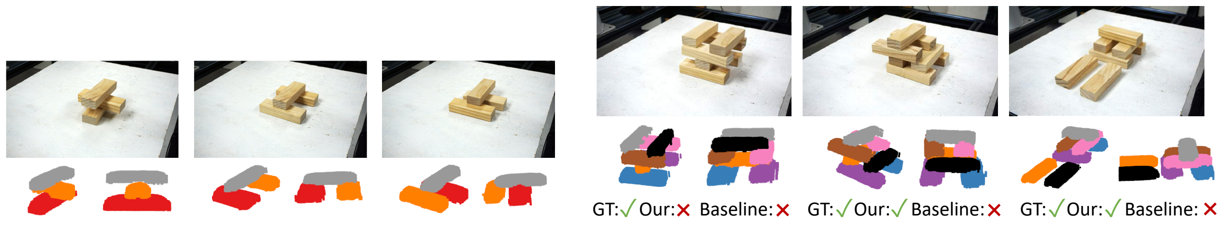


Figure 5.5: *Left:* Example scenes for block unstacking task. *Right:* Test set examples for block unstacking task. The black block shows the block to be removed. Baseline here refers to visual baseline.

5.4.3 Block Unstacking

Block unstacking involves a large amount of geometric and contact based reasoning since the blocks can be arranged in many complex configurations. We formulate the precondition learning problem to predict the stability of a stack of blocks given a particular block to be removed. Figure 5.5 (Left) shows examples with 3 blocks where the grey block can be safely removed from all scenes. In addition to the previous models, we add another GNN based model where graph edges only exist between blocks (vertices) that are closer than a certain threshold (0.1m), thus easing the learning problem. Table 5.2, 5.3 shows results for multiple train-test splits. As seen above, our GNN based models outperform all other methods across all the different scenarios. The 3D CNN based baseline models have a higher wt. F1-score ($\sim$ 0.80) for the 1st-two splits but when tested on objects with different sizes (Table 5.3, Figure 5.6(Left)) they perform much worse. This shows that the 3D CNN based baseline models overfit to the train set objects and cannot transfer to objects with different shapes and sizes. While the DiscreteRel baseline is clearly insufficient for this task. For instance, given a 3-block configuration, with 1 block above and 2 supporting it from below. Although, this configuration has the same discrete representation, the overall stability of this 3-block configuration depends upon the location of the bottom blocks. We discuss this in detail in Appendix 9.1.7.

Also, the real2sim baseline performs poorly on the 1st-two test splits, wt. F1-score ($\sim$ 0.70). This is because blocks are in contact and quite close to each other, their voxel representations are quite noisy

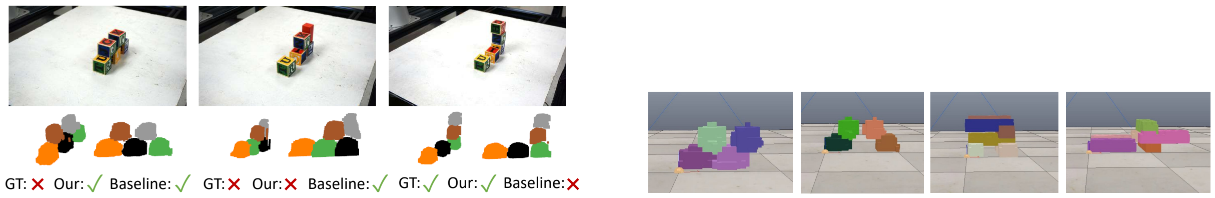


Figure 5.6: *Left:* Test set examples for block unstacking with **different shape and size** than train set. Baseline: visual baseline. *Right:* Example scenes which are unstable but are predicted as stable by the Real2Sim baseline.

e.g. with blocks often embedded into other blocks. These noisy voxel representations when imported into VREP lead to inaccurate predictions. Figure 5.6 (Right) visualizes scenes which are unstable but are predicted as stable in VREP. Interestingly, amongst our models, the relational network (RN) performs similar to the baseline models and much worse than the corresponding GNN models. We believe this is due to the complexity of the reasoning problem. Since we only use one layer of the relational network model, its representation capacity is much less than the corresponding GNN models with 2 layers.

5.5 Ablation Study

We perform an ablation study to understand the effects of different architecture and algorithmic choices in our proposed approach. First, we look at the effects of different loss functions used to train the object relation model. Table 5.4 shows the effect of different loss functions on the precondition learning problem. As seen above, using triplet loss performs better than predictive losses for both food cutting and block unstacking tasks. We believe this happens because the low dimensional embeddings learned using supervised losses alone might not be discriminatory enough for the downstream tasks. In comparison, since triplet loss explicitly forces embeddings to be further apart, it eases the learning problem for the precondition model. Also, comparing Table 5.4 with previous results we see that combining all the losses does outperform using any of the losses individually.

To illustrate the utility of our learned embeddings, we evaluate only using the mean 3D position (similar to [111]) and bounding box of the blocks as input to our sparse GNN model. The bounding box is estimated min-max values for each axes from the voxel representation of the block. Table 5.4 shows that using mean positions alone performs poorly (wt. F1-score: 0.72) while adding object bounds performs better (wt. F1-score: 0.816). However, our GNN model with learned object relations still outperforms them. One reason for this is the sensory noise in the input data, especially since object bounds are sensitive to outliers. More importantly, these results indicate the utility of our learned embeddings which perform well despite sensory noise and limited data. Table 5.4 also shows the utility of the contact based losses for the block unstacking task. Using contacts based losses increase the wt. F1-score on the block unstacking task from 0.80 to 0.824. This is not surprising since the block unstacking task requires inferring contact locations for stability prediction.

5.6 Conclusion

We learn preconditions for manipulation skills by using structured scene representations that decompose scene into its objects and their relations. We propose a novel approach to learn generalizable continuous representations for these object relations. Our approach has several advantages. First, it is grounded in a robot's interaction with objects, and hence we do not assume a fixed set of discrete object relations. Second, simulations provides us access to large set of ground truth data such as contacts distribution which allow us to learn rich representations. Finally, our approach can be directly used for precondition learning in a sample efficient manner.

6 RoboAdapters: Adapting Pretrained Vision Models For Robotic Manipulation

This chapter is based on [Sharma, Fantacci, Zhou, Koppula, Heess, Scholz, and Aytar, 208].

Project page: https://sites.google.com/view/robo-adapters/

Abstract: *Recent works show that large models pretrained on common visual learning tasks can provide useful representations for a wide range of specialized perception problems, as well as a variety of robotic manipulation tasks. While prior work on robotic manipulation predominantly use frozen pretrained features, we demonstrate that in robotics this approach can fail to reach optimal performance, and that fine-tuning of the full model can lead to significantly better results. Unfortunately, fine-tuning disrupts the pretrained visual representation, and causes representational drift towards the fine-tuned task thus leading to a loss of the versatility of the original model. We introduce* ***lossless adaptation*** *to address this shortcoming of classical fine-tuning. We demonstrate that appropriate placement of our parameter efficient adapters can significantly reduce the performance gap between frozen pretrained representations and full end-to-end fine-tuning without changes to the original representation and thus preserving original capabilities of the pretrained model. We perform a comprehensive investigation across 3 major model architectures (ViTs, NFNets, and ResNets), supervised (ImageNet-1K classification) and self-supervised pretrained weights (CLIP, BYOL, Visual MAE) in 3 task domains and 35 individual tasks, and demonstrate that our claims are strongly validated in various settings.*

6.1 Introduction

Pretrained general-purpose vision models, often also referred to as vision foundation models [259], have developed a growing set of perceptual capabilities in recent years. Large-scale vision-language models such as CLIP [179] and ALIGN [79]) are examples of these highly capable general-purpose vision models which have enabled many applications for image generation/editing [182, 193] and image-based dialog [4]. Existing self-supervised pretrained visual models, such as SimCLR [29], BYOL [55] or Visual MAE [61], have also been shown to provide strong initializations for a wide range of visual downstream tasks, and can thus also be considered general-purpose vision models. How can we unlock the power of these models for increasingly novel and challenging control applications?

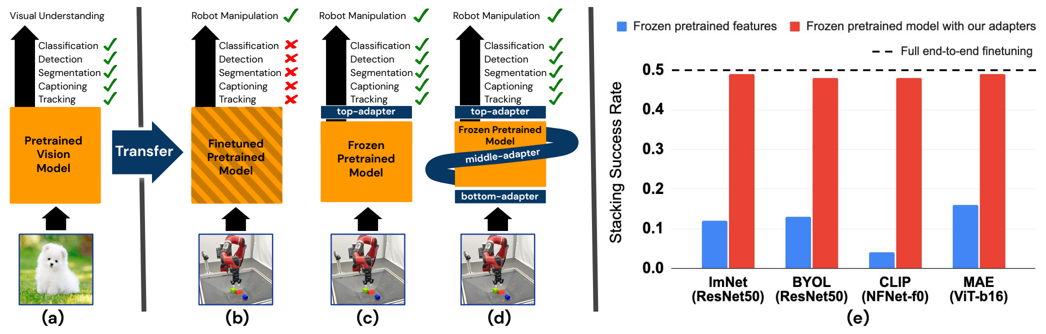


Figure 6.1: **Parameter efficient lossless adaptation**. Existing works adapt preretrained general purpose visual models (a) through full end-to-end fine-tuning as shown in (b), which looses the original capabilities of the model; or adapting frozen pretrained models through top-adapters as shown in (c), which often fails to achieve optimal control performance. However, by introducing additional mid-level and bottom-level adaptation as in (d), we still maintain the existing perceptual capabilities while approaching the full fine-tuning performance as empirically shown in (e) over many network architectures and pretraining methods.

One solution is to add an output head for each control task and fine-tune the entire architecture. However, fine-tuning degrades performance on the original task(s) the model was trained for, and therefore requires maintaining copies of the model for all tasks we wish to concurrently support. This strategy quickly becomes infeasible as we move towards more general and multi-task agents. For instance, embodied agents acting in the real world will end up solving thousands of downstream manipulation tasks. Given limited hardware capabilities of robots keeping separate copies of increasingly large models (e.g. billions of parameters) for a growing set of tasks is unscalable. This is further exacerbated for robot manipulation wherein hardware and tool differences can result in different task configurations which may require different representations.

In this paper our target is to achieve *lossless adaptation*, which we define as adapting the original pretrained model for the new task or series of tasks, while maintaining the original capabilities of the model. To solve the *lossless adaptation* problem we inject additional parameters, i.e. adapters, to several specific locations throughout pretrained architecture. We use similar adapters as in previous non-control settings [67, 184], but carefully insert them at different network locations to improve our off-domain representations for control. We demonstrate that, with a reasonably small cost ($\sim$ 1% of the original model size) of additional parameters scattered throughout the model, we can bridge the performance gap between frozen pretrained features and full end-to-end fine-tuning. Our approach offers close to the performance of full fine-tuning while maintaining all the original capabilities of a pretrained model (the original model definition can co-exist with the new task head, reusing the vast majority of parameters). We show that as the manipulation tasks get harder through complex multi-object interaction and increased level of variation in the randomized initial configurations, the pretrained visual features can't cope with the increased complexity but our parameter efficient adapters can. Overall our contributions include:

- We show that frozen pretrained representations are insufficient to reach optimal manipulation task performance especially for complex manipulation tasks.

- We propose the use of adapters with strong evidence that our adapters can largely close the performance gap between frozen pretrained representations and full end-to-end fine-tuning while adding only a small amount of ($\approx$ 1% of the original model) adapter parameters.
- Comprehensive evaluation of our approach across 3 different manipulation suites (35 individual tasks), 3 major model architectures (ViTs, NFNets, and ResNets) with supervised (imagenet classification) and self-supervised pretraining (CLIP, BYOL, Visual MAE).
- Experiments demonstrating that adapters also help in sim2real transfer. Thus, enabling fixed large pretrained visual models to be directly used for real world manipulation tasks.

6.2 Related Works

The general problem of representation learning for control can be broadly divided into two distinct categories – works that use *in-domain* task data to learn task relevant representations for the underlying task, and on the other hand are more recent works that use *out-of-domain* visual data to learn generally useful representations for control.

In Domain Representation Learning for Control: Within the first broad category the majority of works learn representations that reflect certain invariances that are presumed to be relevant for the downstream task(s) of interest. Prior works show that useful representations can be learned via data augmentations [106, 117], temporal contrastive learning [116], information bottlenecks [166], goal relabeling [268], or via real world priors [83, 84].

Out of Domain Representation Learning for Control: An alternative set of works have emphasized the use of in-the-wild visual datasets for representation learning [95, 171, 219]. They have shown that features learned by large visual models pretrained on common visual learning tasks such as image classification, image inpainting, and contrastive learning can be surprisingly effective for downstream control tasks. Many of these works utilize the large scale pretrained CLIP model [51, 95, 219], while other works also show the effectiveness of features trained on ImageNet [171, 205], or using temporal data from Ego4D [158]. Our work is similar in spirit to these works, i.e., we focus on models trained on large scale out-of-domain data. However, in contrast to prior work which solely shows the effectiveness of the extracted off-the-shelf representations, we aim to adapt those representations for better downstream performance.

Transfer Learning: Adapting a given large model for different downstream tasks has been widely studied in the literature [26, 27, 80, 126, 127, 136, 184, 236]. Within the vision community [184] introduced adapter modules to learn a single representation across multiple domains. [126] used similar adapters for few-shot image classification in new domains. However, previous works within vision mostly use adapters for classification tasks and the pretraining task is also image classification. By contrast, we focus on control tasks which are significantly different than image classification. In addition to task-differences our work also includes significant domain shifts, e.g., non-canonical camera views, moving robot arm, heavy occlusion, textureless objects. As we show, unlike previous works, this requires require carefully inserting adapters in different sections without tremendously increasing the parameter count. Adapter modules have also been explored in the language community. [67] adapt the BERT model for different tasks, while

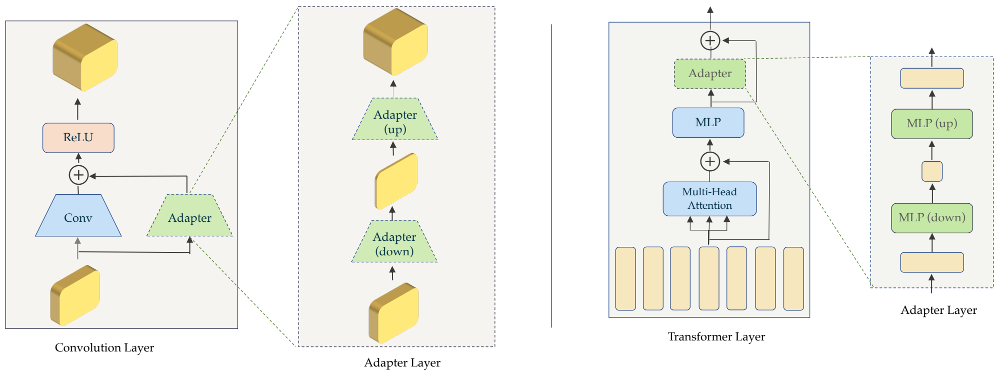


Figure 6.2: Adapter layers used for convolution based (Left) and transformer based (Right) architectures. For both scenarios we use a bottleneck design.

[127] show that optimizing small amount of task-specific vectors (prefixes) can often match the full fine-tuning performance [133]. Another aspect of transfer learning is continual or lifelong learning wherein learning involves a sequence of tasks [5, 183, 192, 202, 218, 256], but wherein the original model is distilled for new tasks and thus not necessarily perfectly preserved. Our work is similar to the former set of works – we keep the large pretrained model *frozen* and improve it using light-weight adaptation to extract the final representation for control.

6.3 Approach

Our main aim is to use fixed pretrained visual models but adapt their representations for improved downstream control task performance. To achieve this we use parameter efficient adapter modules that can be inserted in appropriate locations throughout the deep network. These non-pretrained adapter modules are the only set of parameters that are updated during downstream policy learning.

A common approach to use fixed pretrained visual models is to attach a learned policy head (i.e. top-adapter) and train for the downstream control task [158, 171]. However such an approach cannot adjust the low-level perception and mid-level abstraction for the downstream control task. By contrast, other works have shown that adapting the initial layers (i.e. bottom-adapter) can be important for transferring across large domain-shifts (i.e. sim2real [76], images to line-drawings [7]). One can also inject mid-level adapters as demonstrated in many non-control settings [67, 184]. We adapt these works for the control setting and show how carefully inserting adapters in different sections of the network leads to improved control task performance without any loss of information.

6.3.1 Adapter Modules

Adapter modules are light-weight neural modules that can be inserted at different layers of a pretrained deep neural network. Prior works have explored adapter modules for transfer learning, wherein adapter modules are inserted at each layer of a pretrained deep network and only these adapters are updated at the fine-tuning stage [67]. Overall, adapter modules have two important properties, 1) they are *lightweight*,

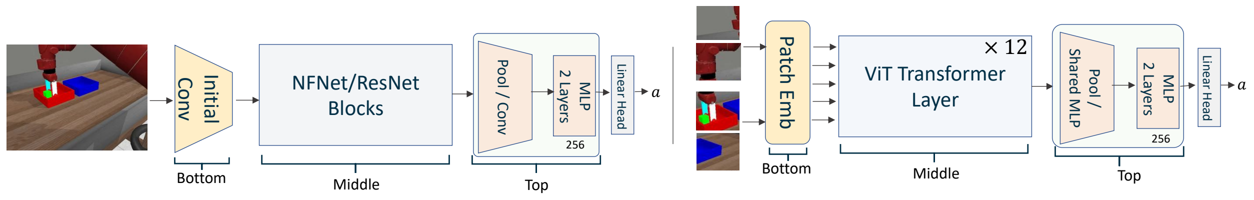


Figure 6.3: Different locations to insert adapter modules for convolution (Left) and transformer (Right) models.

i.e., they have fewer parameters compared to the original network, and 2) they keep the *initialization* provided by the pretrained deep network. Below, we provide a short discussion on how adapter modules achieve these properties.

Prior works limit the number of parameters in the adapter modules by either using 1×1 convolutions [184] or via bottleneck architectures for transformers [67]. In our formulation, we use a bottleneck architecture for both convolution- and attention-based architectures, as well as utilizing 1×1 convolutions for the former. Specifically, for a given input $x \in \mathbb{R}^{n \times f}$ where n is the number of samples, f is the input feature dimensionality, adapter outputs are parameterized as a combination of down- (d) and up-projection (u) weights: $x' \leftarrow W_A^u\big(h\big(W_A^d x\big)\big)$, where h is some non-linearity, $W_A^d \in \mathbb{R}^{f \times f'}$ and $W_A^u \in \mathbb{R}^{f' \times f}$ are the adapter (A) weights, f' is the bottleneck feature size. Since $f' << f$, the adapter modules utilize a very small number of parameters in comparison to the original pretrained network. Importantly, we use the above formulation for both convolutional and transformer based architectures – for convolutional architectures we downproject across the channel (feature) dimension, while for transformer based architectures we downproject along the feature dimension for each patch. Figure 6.2 visualizes the adapters used in our work.

To fully utilize the initialization provided by the pretrained weights, we initialize the adapter modules with weights close to 0 and add skip connections when inserting the adapter modules into the pretrained network. This ensures that adapter modules have no effect at initialization and thus the pretrained network's initialization remains intact. Overall, for a input x, the output for a pretrained model's layer with an adapter is $x' \leftarrow g_{\text{pretrain}}(x) + g_{\text{adapter}}(x)$. Finally, we note that the adapter modules can also be added serially, i.e., directly after the output of a pretrained layer, in which case g_{pretrain} can be viewed as an identity function. Finally, similar to previous works [37, 67, 176] we also add offsets to the normalization parameters used in different architectures (e.g. layernorm [8] for vision transformers).

6.3.2 Visual Adapters for Control

We first discuss why adapting out-of-domain representations for control is different from previous usage of adapters for language and visual understanding tasks.

Task Differences: Prior works often utilize adapter modules for a broadly similar set of tasks in a given domain. For instance, text classification tasks [67], neural machine translation tasks [178], or few-shot visual classification tasks [126]. Additionally, the pretrained model used in these tasks (e.g. BERT [40] or ImageNet pretraining) is strongly correlated with the downstream tasks. By contrast, there exists much larger *task differences* in our work – the visual pretraining task (e.g. image-inpainting/classification) is vastly different from downstream robot manipulation (continuous action prediction). While visual classification tasks mostly require semantic features (i.e. what is in an image), manipulation tasks require actionable features ("what" and "where" are the objects to predict actions) [43, 189]. These actionable features need both semantic as well as spatial features (e.g. being able to localize objects of interest in space).

Domain Shifts: Few works in language understanding use adapters for *domain shifts* [265] (even [265] use unlabelled target data to pretrain adapters). While works in visual understanding do involve domain shifts, e.g., train on ImageNet, Birds and transfer to MS-COCO, CIFAR-10 [126]. However, such shifts are very different from our work – robot manipulation data consists of table-top settings, simulation images, moving robot arm, heavy occlusion, which is very different from object-centered images of ImageNet. Based on above differences we next discuss how and where to efficiently insert adapter modules to improve upon "off-the-shelf" representations produced by the fixed pretrained network.

While we can add adapter modules through all layers of the pretrained network, such a choice is highly parameter inefficient and redundant especially for large networks with many layers. We coarsely categorize the network layers based on their functional forms as *bottom*, *middle* and *top* sections as visualized in Figure 6.3. Next, motivated by the above discussion on visual adapters for control, we provide intuitive reasoning for injecting adapters in each section. Importantly, as we show empirically, unlike previous works [67, 126, 184] using adapter parameters at each of these network sections is important for task performance.

The *bottom* layer directly uses the raw images as input. In scenarios, where there is a mismatch between the downstream task's image observations and the pretrained bottom layer feature statistics, the downstream task performance can be sub-optimal. As discussed above, such scenarios are common for downstream manipulation tasks, since there exists a significant domain gap between the data distribution of pretrained vision models (often in-the-wild data) and standard table-top settings with much closer and non-canonical camera views.

The *middle* category, which contains most of the fixed pretrained network ($\approx 90\%$) weights, is used to extract the appropriate input abstraction. However, these network weights are trained on visual learning tasks which often focus on semantic understanding (e.g. image classification) instead of spatial and causal understanding which are important for control. Nevertheless, earlier layers of the network are known to capture useful invariances [171, 262]. Hence, *sparsely* inserting adapter layers through the pretrained network can allow us to better adapt "off-the-shelf" representations for downstream manipulation tasks. Finally, we note that the output of the middle category is a set of spatial features for conv-nets and patch features for ViTs.

The *top* category uses the spatial representation from the middle category as input and outputs the robot action. This high dimensional spatial representation (size $\approx 20K$) is converted into a smaller representation (size $\approx 2K$) either via average/max pooling or by down-projecting using 1×1 convolutions or a small shared MLP. Finally, this smaller representation can be used to directly output the action using a linear policy head. While a linear head is sufficient in settings where there is strong task alignment [29], given the large difference in the pretraining and downstream tasks, additional top layer adaptation helps further improve performance. We note that most prior works that use fixed pretrained features [158, 170]

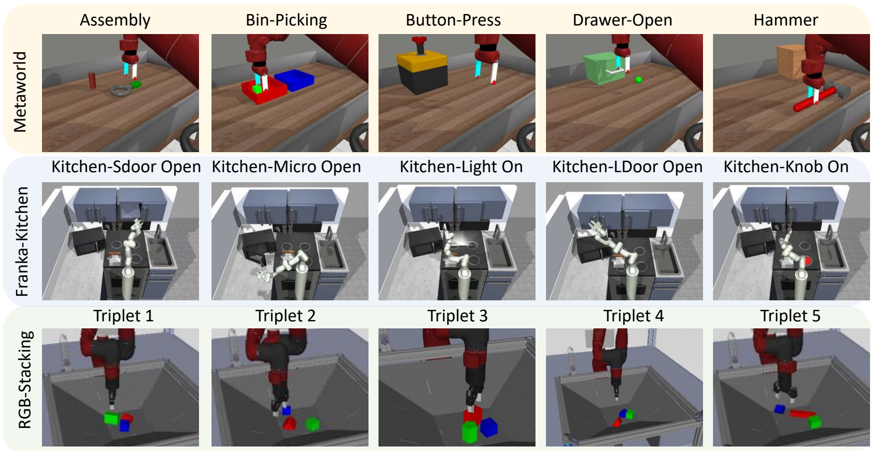


Figure 6.4: Different environments we evaluate our approach on. For Metaworld and Kitchen suites we frollow the setup from [158] including the same set of demonstrations. For RGB-Stacking suite we use Skill Mastery setting [119].

also utilize such top layer adaptation. We discuss implementation details for specific architectures in Section 6.4.2.

Training Adapters for Control: We use behaviour cloning to learn task policies from demonstrations. We use euclidean loss to optimize adapter parameters.

6.4 Experimental Setup

We evaluate our use of adapters for large pretrained visual models across three different environment suites each with increasing task complexity and across three different network architectures. We elaborate on each of these further below.

6.4.1 Manipulation Tasks

In this work, we consider Metaworld [258], Franka-Kitchen [56], and RGB-Stacking task suites [119]. Figure 6.4 visualizes the tasks considered from each of these task suites. Both Metaworld and Kitchen have been used previously [158] to evaluate fixed "off-the-shelf" pretrained visual representations. Hence, for both suites we use the same environments and demonstrations. Overall, we use 5 different environments from the Metaworld and Kitchen task suites. For each environment we use 3 different camera configurations as provided by [158]. Additionally, similar to previous work we use 25 demonstrations for each environment and train a separate policy for each environment and camera configuration.

While both Metaworld and Kitchen suites consider many tasks, each task often has a very narrow state-space distribution. For instance, Kitchen tasks use fixed object positions while MetaWorld tasks have limited position variance only. Hence, we also evaluate our approach on a much more challenging RGB-stacking suite [119] (Figure 6.4 bottom). The RGB-stacking tasks involve three objects colored Red, Green, and Blue and the goal is to stack the red object on top of blue object. These tasks randomize object positions and orientations in a basket and thus result in a large initial state-space distribution. Further, since these tasks consider a diverse range of shapes, stacking requires precise control and cannot be achieved simply by dropping one object onto the other. To obtain demonstrations for these tasks, we initially train an RL policy and collect 100,000 demonstrations from it. We use the *Skill Mastery* setting from [119] for evaluation.

6.4.2 Network Architectures

We evaluate the effectiveness of adapters for manipulation tasks in the context of three different network architectures. Specifically, we use normalization free networks (NFNet) [19], residual networks (ResNet) [63] and vision transformers (ViT) [42]. Among the different architectures within each category we use *NFNet-f0*, *ResNet-50* and *ViT-B/16*. In addition to imagenet pretraining across all three architectures, we also evaluate using ALIGN [79] for NFNet, BYOL for ResNet [55] and masked auto-encoder (MAE) for ViT [61].

Adapter Parameterizations: As noted previously in Sub-Section 6.3.2 we use *bottom*, *middle* and *top* adapters. We discuss the specific parameterizations for each of these adapter types. Table 6.4 provides an overall adapter parameter count. We provide detailed parameterization in Appendix 9.4.1.

Bottom: For NFNet and ResNet architectures we use the initial convolution, while for ViT we use the initial patch embedding as the only bottom set of layers. We add 1 adapter for this bottom layer.

Middle: For middle adapters we use 4 and 6 adapters for the convnet and transformer based architectures respectively. For the convnet based architectures we add each adapter to the first convolution in each block group except the last group, where we add it at the end. While for the transformer based architectures we apply them at layers $\{0, 1, 5, 6, 10, 11\}$.

Top: We project spatial features from the middle layer onto a lower dimensional space. To process these features we *optionally* add 2 MLPs each with 256 parameters, we refer to these as top adapters.

Evaluation: We evaluate our approach using 40 rollouts from the BC learned policy. We use mean success rate of the final policy as our metric. When providing task suite metric we average the mean success rate across all environments and camera configurations. We also note that some previous works evaluate the BC policy at fixed training step intervals and use the maximum over the mean success rates at each interval as the metric. However, using max as a statistic is not robust and is easily influenced by outliers. While comparing with previous works we use the max statistic, however for all other results we use the mean. Training details and hyper-parameters are in Appendix 9.4.1.

6.5 Results

With our experiments we aim to show the following: 1) Using fixed pretrained representations often leads to sub-optimal task performance as compared to directly adapting the representation for the downstream task, especially for more challenging manipulation tasks. 2) Inserting small adapter modules into a fixed pretrained network (i.e. lossless adaptation) is sufficient to achieve optimal manipulation task

	Metaworld	Franka-Kitchen	RGB Stacking
Fixed Pretrained Feat. - ImNet (R3M) [158]	0.66	0.30	-
Fixed Pretrained Feat. - MoCo-345 (R3M) [171]	0.64	0.38	-
Fixed Pretrained Feat. - R3M (R3M) [158]	0.78	0.56	0.30
Fixed Pretrained Feat- (Ours - ImNet)	0.62	0.48	0.28
Full Finetune (Ours)	0.98	0.72	0.70
Fixed Pretrained Feat. - (Ours ImNet - mean succ)	0.44	0.34	0.14
Full Finetune (Ours - mean succ)	0.91	0.57	0.49

Table 6.1: Success rate comparison using fixed pretrained features (using pretrained large vision models) with methods that update the pretrained visual features on the downstream manipulation task.

	Metaworld			Franka-Kitchen			RGB Stacking		
	Pretrain Feat.	Adapters	Full FT.	Pretrain Feat.	Adapters	Full FT.	Pretrain Feat.	Adapters	Full FT.
NFNet	0.44	0.82	0.94	0.12	0.39	0.42	0.14	0.45	0.47
ResNet	0.39	0.80	0.88	0.12	0.24	0.25	0.14	0.48	0.49
ViT	0.19	0.78	0.84	0.15	0.26	0.25	0.18	0.49	0.48

Table 6.2: Mean success rate comparisons between using *fixed* pretrained features, adapters and full fine-tuning across all three different environments with three different architecture choices.

performance across a wide range of tasks and network architectures. 3) Adapter modules are parameter efficient as well as robust to different initializations of the fixed large pretrained vision model (e.g. via different visual pretraining strategies). We look at each of these points in the following subsections.

6.5.1 Fixed Pretrained Features vs Adapter Representations

Fixed Off-the-Shelf Representations: In the first part of our experiments we show that while fixed off-the-shelf representations (without any adaptation) are useful, they can be highly sub-optimal for the given downstream task. To show this we compare the fixed representations extracted using pretrained weights (Pretrained Feat.) obtained via supervised imagenet pretraining and compare them with full fine-tuning (Full FT). Table 6.1 compares these across all task suites. For a fixed comparison with previous works Table 6.1 reports results for the ResNet-50 model, since previous works only evaluate the ResNet architecture. As seen in Table 6.1, fixed off the shelf-representations are comparatively much worse across all environment suites. For Metaworld, Kitchen and RGB-Stacking suites the *relative* change in performance is around $\approx$ $20\%, 30\%$ and 100% respectively. Also, for the RGB-stacking suite the mean performance is much lower 14%, which shows that fixed pretrained representations become significantly less effective for challenging manipulation tasks.

Lossless Adaptation of pretrained Visual Features: We now show that our proposed adapters can match full-fine-tuning performance for downstream manipulation tasks without losing any existing information. Table 6.2 compare full fine-tuning (Full FT.) approaches with our adapters (Adapters) as well as fixed pretrained representations (Pretrained Feat.) We note that for these and future results we report metrics using the more robust mean statistic. Additionally, for task suites with limited state-space distributions, *i.e.*, Metaworld and Franka-Kitchen, we avoid using any proprioceptive information (see Appendix 9.4.3 for results with proprioceptive). This allows us to robustly verify the visual representa-

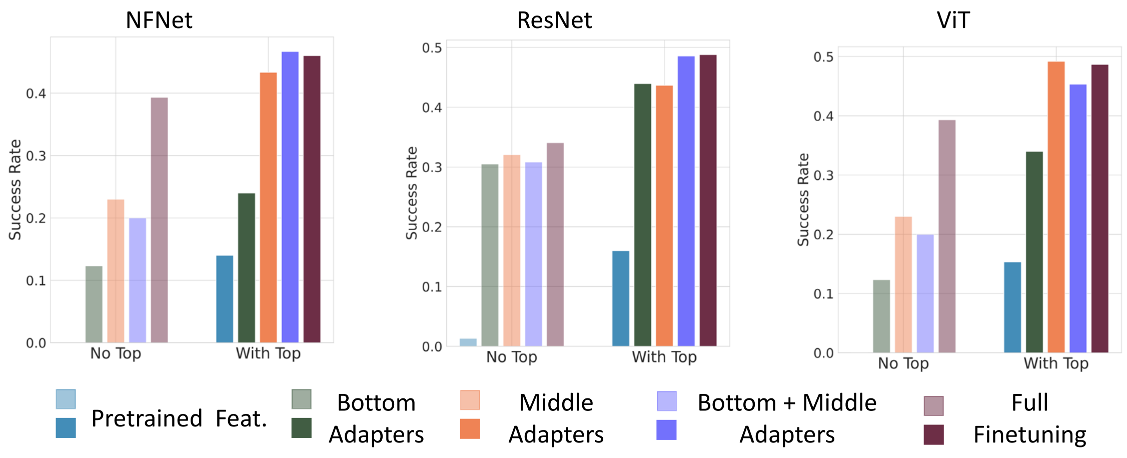


Figure 6.5: Ablation results on the RGB-Stacking environment for 3 different network architectures.

tion and avoids any proprioceptive information leakage which can allow to solve the task even without using the visual features.

Table 6.2 shows the results for each task suite and network architecture combination. For the adapter results we report results with bottom, middle and top adapters. As before, for a fair comparison we use top adapters for all other approaches. As seen in the above table, our parameter efficient adapters can closely match full fine-tuning performance across all environment suites and architectures. Most noticeably, for both Franka-Kitchen and RGB-Stacking tasks, our use of adapters is able to exactly match (average difference in performance $< 2\%$) the performance derived from full fine-tuning (Full FT). While there exists a slightly larger gap for the metaworld environments – average performance difference $\approx 6\%$. However, compared with directly utilizing the fixed pretrained features, we see a huge performance increase of around 30% averaged over all tasks and architectures. In addition to the average results, Table 6.2 also shows that there exists a performance increase across all architectures. Our results clearly show that the features derived from visual pretraining are quite useful and inserting few additional adapters allows us to achieve close to optimal performance without sacrificing any of the previously learned abilities of the pretrained vision models.

6.5.2 Effects of Adapter Locations & Different Pretrained Representations

We investigate the effect of inserting adapters in each of the different network layers (see Subsection 6.3.2). Figure 6.5 shows results for inserting adapters in each network layer for RGB stacking suite (see Appendix 9.4.3 for results across all suites). We split each network plot above into two parts – 1) without using top layer adapters (i.e. directly using a linear policy head), and 2) using a top layer adapter (i.e. using 2 additional MLPs before the linear policy head).

From Figure 6.5 we see that *without* top layer adapters (greyed out left plots) the performance for all methods decreases – more so for fixed pretrained features. As seen above, top adapters are crucial, e.g., not using top layer adapters almost *halves* the control performance across all different architectures. Figure 6.5 also shows that bottom adapters alone can give a significant performance boost (almost $2\times$ than Pretrained

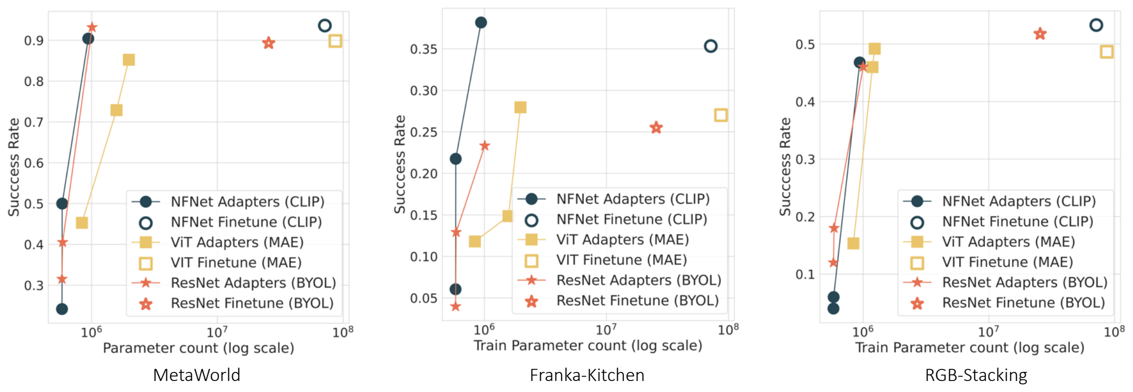


Figure 6.6: Results with different pretraining initializations (for 3 different models) all 3 environments – NFNet with CLIP, ResNet with BYOL and ViT with MAE. *Bottom Left:* points plots performance of fixed pretrained features with top adapters. *Top Right:* points plot full fine-tuning performance (with top adapters). *Solid Lines:* indicate adapter performance with adapters first added to bottom and then middle layers.

Feat. with top adapters). Since bottom adapters for conv-net architectures have very few parameters (a few thousand see Table-6.4), this performance increase is not merely a function of more parameters and thus shows the need for better adaptation of pretrained visual features. Overall, best performance is often achieved when using all top, middle and bottom adapters together. Since previous works [126, 184] only consider adapters in middle sections, our results show the benefits of bottom/top adapters for control tasks.

Adapters with Different Pretrained Representations: We now show that our proposed adapters give similar benefits with pretrained weights obtained from vastly different pretraining (pretext) tasks. For NFNet we use CLIP pretraining [179], for ResNet we use BYOL [55] and for ViT we use masked auto-encoder (MAE) [61]. Figure 6.6 plots the result for each of these architectures across all three task suites. The X-axes shows the number of train parameters (see Table 6.4). The *bottom left* points in each of the above plots indicate the performance of fixed pretrained features. While *top right* points show full fine-tuning's performance. On the other hand, the solid lines indicate the performance improvements on inserting bottom and middle adapters (top adapters are used for all approaches). As seen in Figure 6.6, for all tasks and pretrained weights adapters are able to closely match the performance of full fine-tuning approach, while only registering a very marginal increase in the number of trainable parameters.

Additionally, comparing MetaWorld results in Figure 6.6 (Left) and Table 6.2 we see that while there exists a minor gap between adapters and full-FT with imagenet-supervised weights $\approx 6\%$, this gap reduces significantly for self-supervised pretraining weights. Somewhat similar results on the benefits of MAE features for control were also observed in [251].

Additionally, similar to some previous works [180], we also find that CLIP pretrained weights (top adapters only) can perform poorly. For instance, they get $< 5\%$ on RGB-stacking tasks. However, using adapters the performance significantly improves $\approx 50\%$ and closely matches full fine-tuning performance. Moreover, the adapted representations match the performance of models known to be performant (e.g. MAE [251]).

	Triplet 1	Triplet 2	Triplet 3	Triplet 4	Triplet 5	Average
NFNet - Pretrained Feat.	0	0	0	0	0	0.0
NFNet - Scratch	0.02	0.02	0.01	0.16	0.04	0.05
NFNet - Adapters	0.18	0.14	0.16	0.47	0.23	0.24
NFNet - Full FT.	0.22	0.36	0.15	0.76	0.26	0.35
NFNet - Pretrained Feat. (DR)	0.00	0.02	0.0	0.06	0.03	0.02
NFNet - Adapters. (DR)	0.38	0.40	0.38	0.88	0.64	0.53
NFNet - Full FT. (DR)	0.40	0.36	0.32	0.91	0.62	0.52
ViT - Adapters	0.04	0.08	0.04	0.24	0.12	0.10
ViT - Full FT.	0	0	0	0	0	0

Table 6.3: Sim2Real results for RGB-Stacking Task with and without using any visual domain randomized (DR) data for learning the manipulation task policy.

	Bottom	Middle	Top	Full
NFNet-f0	0.5K	0.4M	0.6M	71M
ResNet-50	6.9K	0.4M	0.6M	25M
ViT	0.74M	0.4M	0.9M	86M

Table 6.4: Number of parameters to be learned for different adapters as well as full fine-tuning.

6.5.3 Sim2Real Results

Finally, we investigate if large scale visual pretraining combined with our use of adapters can allow for *sim2real transfer*. Prior works that utilize fixed pretrained vision models for real robot tasks often only evaluate on tasks requiring simple motions (reach/grasp) and almost always train the policy on real robot data [158, 219]. By contrast, we show results for sim2real transfer, i.e, we use no extra real-world robot data. We use the more challenging RGB-stacking suite for evaluation. We evaluate sim2real transfer both *with* and *without* using any visual domain randomization data. We report results using 100 rollouts for each policy and object triplet.

Table 6.3 shows results for one conv- (NFNet) and transformer- (ViT) architecture. From the above results we see that ViT based policies perform much more poorly compared to NFNet policies. For instance, a fully fine-tuned ViT policy is completely unable to solve any task. We believe one reason for this is the high learning capacity of transformer based models which allows it to quickly learn on the given task and lose any prior information [249], thus making real world transfer challenging. However, using adapters instead of fine-tuning is able to achieve non-zero performance. While the average performance is poor 10%, it is able to achieve 24% success on the easier setting (triplet 4). Table 6.3 further shows that NFNet based policies perform much better than ViTs across different training settings. For full fine-tuning and adapter approaches NFNet policies can achieve 35% and 24% success rate, while training from scratch only achieves 5% and fixed pretrained features do not result in any successes. Finally, we also evaluate the NFNet policies using visual domain randomization data (DR rows). These policies show much superior performance $\approx$ 53% for our adapters and full fine-tuning, and closely match the policy performance in simulation.

6.6 Conclusion

In this work we propose the lossless-adaptation problem, i.e., we aim to adapt representations from large-scale pretrained vision models for close to optimal manipulation task performance. We show that using fixed representations by solely using top-adapters (as is common) can fail to achieve optimal task performance especially for challenging manipulation tasks. To solve this we propose our parameter efficient adapters. We show that that inserting these adapters at appropriate network locations can achieve close to optimal downstream task performance (closely matching full fine-tuning performance). We show that our results hold across 3 different manipulation suites, 3 different network architectures and multiple different pretrained weights. Further, we show that adapters also allow for sim2real transfer, all while maintaining the pretrained network's original capabilities.

7 Multi-Resolution Sensing for Real-Time Control with Vision-Language Models

This chapter is based on [Saxena, Sharma, and Kroemer, 196].

Project page: https://mohitsharma0690.github.io/multi-res-real-time-control/

Abstract: *Leveraging sensing modalities across diverse spatial and temporal resolutions can improve performance of robotic manipulation tasks. Multi-spatial resolution sensing provides hierarchical information captured at different spatial scales and enables both coarse and precise motions. Simultaneously multi-temporal resolution sensing enables the agent to exhibit high reactivity and real-time control. In this work, we propose a framework for learning generalizable language-conditioned multi-task policies that utilize sensing at different spatial and temporal resolutions using networks of varying capacities to effectively perform real time control of precise and reactive tasks. We leverage off-the-shelf pretrained vision-language models to operate on low-frequency global features along with small non-pretrained models to adapt to high frequency local feedback. Through extensive experiments in 3 domains (coarse, precise and dynamic manipulation tasks), we show that our approach significantly improves (2× on average) over recent multi-task baselines. Further, our approach generalizes well to visual and geometric variations in target objects and to varying interaction forces.*

7.1 Introduction

Performing robotic manipulation tasks in the real world often requires using sensing modalities at different *spatial resolutions*. For instance, for peg-insertion, the robot can use a statically-mounted third-person camera (low spatial resolution or global information) to reach close to the hole, use a wrist-mounted first-person camera for finer alignment, and finally use proprioception and force-feedback for insertion (high spatial resolution or local information). Additionally, each sensing modality can be utilized at a different *temporal resolution*. For example, for coarse quasi-static subtasks ("reach hole"), using third-person camera images at a *low frequency* can be sufficient. However, finer reactive subtasks ("insert peg"), might require *high-frequency* force-torque feedback. Based on this insight, we propose a multi-resolution (spatial and temporal resolution) sensor fusion approach for coarse quasi-static as well as precise reactive manipulation tasks.

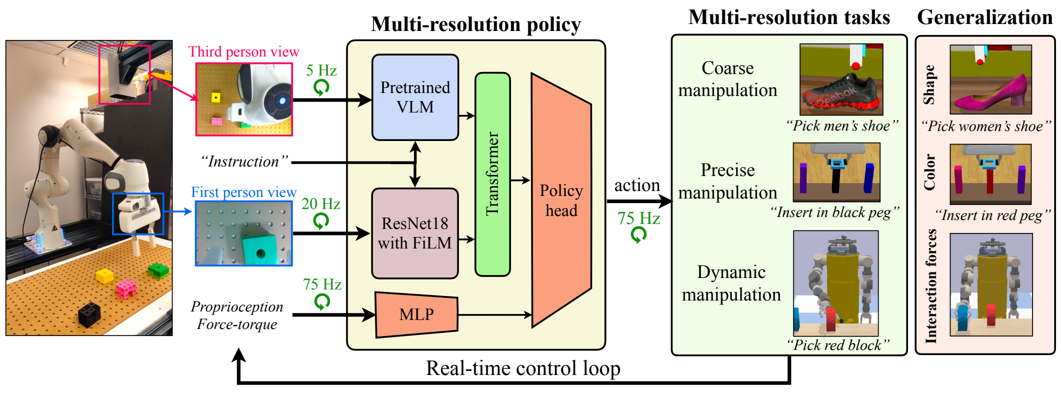


Figure 7.1: Our proposed approach uses sensing at different spatial and temporal resolutions for real time control of coarse, precise and dynamic tasks while enabling generalization to novel visual features and interactions.

Multi-resolution sensor fusion can enable generalization to novel visual-semantic targets. For instance, by utilizing global information from third-person camera images only for coarse localization and relying on local information from in-hand cameras and force-torque feedback for finer motions, the policy can learn to generalize to novel objects. Previous approaches to learning generalizable policies either require extensive data collection [20, 73, 185] or rely on pretrained models [4, 79, 123, 179] for policy adaptation [219]. However, such approaches typically utilize a single sensory modality, while others that incorporate multiple sensors do not prioritize generalization [220]. In our work, we avoid extensive data collection and instead leverage pretrained vision-language models in our multi-resolution approach to learning generalizable language-conditioned multi-task policies.

Although pretrained vision or vision-language models (VLMs) provide impressive generalization capabilities and enable learning language-conditioned multi-task policies, using large VLMs can have certain disadvantages. First, given their large size (e.g. Flamingo has 80B parameters [4]), they have slow inference which makes them unusable for real-time closed-loop control which is necessary for reactive tasks. Second, since pre-trained models are often trained on out-of-domain data, using them to solve in-domain manipulation tasks (especially precise tasks) may require finetuning [207]. However, task-specific finetuning can make models less robust with reduced generalization [249].

To overcome the above challenges of utilizing large pretrained VLMs for real-time control of reactive tasks, we propose a framework that incorporates different capacity networks (that operate on different sensing modalities) at different frequencies. Specifically, we use large pretrained VLMs with slow inference at a lower frequency while small networks with fast inference at a higher frequency. Our low-frequency pretrained VLMs operate on statically mounted third-person views and can provide global coarse feedback (such as approximate object locations) that is usually only needed at a low rate. On the other hand, we propose using small trained-from-scratch models with first-person camera views and force-torque data to obtain the high-frequency fine-grained feedback necessary to perform precise and reactive tasks. Further, to overcome the challenge of loss in generalization when finetuning pre-trained VLMs, we *freeze* the pretrained VLMs to avoid losing their robustness and maintain their generalization abilities. Overall main contributions include:

- a framework for learning generalizable multi-task policies that incorporates multiple sensory modalities to capture global to local spatial information,
- combine sensor modalities at different frequencies to avoid bottlenecks and enable reactive control which we show empirically is essential for dynamic tasks,
- comprehensive experiments across 3 domains (and 2 real-world tasks) that include coarse, precise and dynamic manipulations tasks, and
- effective generalization across semantic task variations in both simulation and real-world.

7.2 Related work

Vision-Language Pretrained Models for Robot Manipulation: Many prior works combine vision and language for robotic tasks. While early works focus on tabula-rasa learning [144, 240, 246], more recent works, use pretrained large language models (LLMs) and show efficient learning and improved generalization for robotics tasks [2, 51, 128, 132, 226]. Many recent works also combine large general-purpose pretrained vision or vision-language models (VLMs) [4, 179, 225] for manipulation [148, 149, 158, 171, 207, 219, 234, 250, 266]. Our work is more closely related to these latter works in that we also use pretrained VLMs for robot manipulation. Among these works, many works only use language for task-specification and do not focus on the generalization provided by pretrained models [148, 149]. Additionally, other works adapt the pretrained representation for the downstream task [207, 208, 250]. However, as we show empirically, such updates lead to representation drift and a loss of robustness for the pretrained general-purpose VLM. Hence, we propose not updating the pretrained representations. While [219, 234] use frozen VLMs, [234] only uses pretrained VLM as an open-world object detector to get pixel targets for the task at the first episode step. On the other hand, [219] uses the pretrained VLM with templated pick-and-place actions for manipulation. By contrast, we use VLMs in our multi-resolution framework with continuous feedback for reactive manipulation tasks.

Multi-Spatial Resolution for Robot Manipulation: Many prior works use multiple sensor modalities for robot manipulation, wherein each modality operates at a different spatial resolution. For instance, prior works often combine visual (low spatial resolution) and proprioceptive (high spatial resolution) feedback [90, 120, 122], use wrist-mounted cameras for visual servoing [107, 163, 257] or for contact-rich manipulation tasks [82, 152, 231, 244], while other works focus on combining vision and haptic sensing [23, 45, 121, 124]. Our work is similar to the first set of works i.e. we use both third person and first person cameras for precise manipulation. However, unlike most prior works [231, 244] which focus on single-task settings, we focus on multi-task settings and fuse multiple sensing modalities at different resolutions.

Multi-Temporal Resolution for Robot Manipulation: Learning reactive policies requires the robot to operate at high frequencies. Some recent works in robot manipulation focus on learning policies at different temporal resolutions. For instance, [160] decompose a manipulation task into different phases (e.g. visual reaching phase and tactile interaction phase) and learn separate policies for each phase as well as a blending policy. While [227] avoid the discrete formulation of an MDP and instead learn a continuous differential equation [50, 97] to model the low resolution features. By contrast, we use the

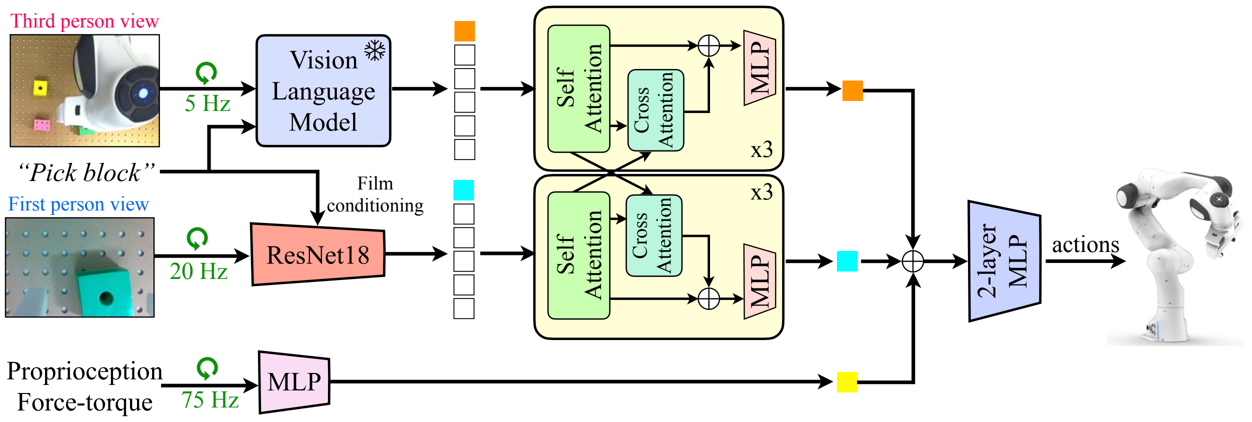


Figure 7.2: Overall architecture: Global low frequency information is extracted from third-person camera images using slow inference networks, local high frequency information is extracted from first-person camera images and proprioceptive, force-torque feedback using fast inference networks. These sensing modalities are then fused at different frequencies to enable real time high frequency control.

discrete formulation and instead of decomposing policies into different phases we reuse features from low-resolution signals while operating at a high temporal resolution.

Dynamic Reactive Manipulation: Many prior works in robot manipulation focus on quasi-static tasks [20, 51]. However, there has been increased interest in solving tasks that are reactive and dynamic in nature [154, 195, 216]. Previous works focus on explicitly learning the dynamics [195] or using analytical models [216, 221] of such systems for achieving reactivity. These works often assume access to the ground truth object pose and are limited to a single-task setting. In our work, we learn how to perform such dynamic and reactive tasks using visual inputs in a multi-task setting.

7.3 Proposed Approach

In this section, we discuss our approach for learning a generalizable language-conditioned multi-resolution multi-task policy for precise and reactive manipulation tasks. Below, we provide details on how we utilize different sensing modalities and then delineate our training/inference and discuss how our approach enables real time control for reactive tasks while generalizing to novel tasks.

7.3.1 Multi-Resolution Architecture

Figure 7.2 shows the architecture of our multi-resolution approach. Our model takes as input multiple sensing modalities with different spatial resolutions, i.e., statically-mounted third-person camera view, first-person camera view and high frequency force-torque feedback. Each input is first processed separately before being fused together at different temporal resolutions to output high frequency robot actions. Below we expand on each component of our architecture.

Low-Spatial Resolution Model: We use a low-spatial resolution sensor (third-person camera) to provide global task information to our agent. We use pretrained visual-language models to extract this global information from third-person views as well as to enable language-conditioning in a multi-task setting. Such pretrained models enable generalization to novel semantic features such as new objects or novel language commands. However, to ensure the pretrained model maintains its robustness we keep it *frozen*. However, using large VLMs to extract this generalizable global information comes with the drawback that the inference speed is very slow ($\approx$ 5Hz). We experiment with two models CLIP [179] and MDETR [91] (language-conditioned DETR [26]), which use image-level and object-level information respectively.

High-Spatial Resolution Model: To ensure reactivity in the face of slow inference of pretrained VLMs, we use a smaller non-pretrained vision model (ResNet-18) [63] to process the first-person camera view at a higher frequency ($\approx$ 20Hz). This view provides us with high-resolution local spatial information. To provide appropriate task-context to the first-person view we use small FiLM layers [176] for language conditioning. We train this model from scratch with augmentations (explained in the next paragraphs) to extract local spatial features that are useful for precise tasks. While using a small vision model enables faster processing it can still be insufficient for some highly dynamic tasks. Hence, we process the force-torque feedback and proprioceptive information at a much higher frequency ($\approx$ 75Hz) using a small linear layer.

Multi-Resolution Sensor Fusion: We combine local and global sensing information (spatial resolutions) mentioned above at different temporal resolutions based on the capacities of the respective networks. Specifically, we reuse features (network activations) from lower frequency (third-person and first-person views) networks to match the frequency of the highest frequency (force-torque feedback) network. Doing this ensures that the policy network outputs actions at a high frequency (equal to the frequency of the force-torque feedback network), thus enabling real-time control.

In addition to temporal-sensor fusion we also spatially fuse local and global sensing information, i.e, we fuse information extracted from third-person views with first-person view information and vice-versa. We achieve this using two small camera-specific transformers together with cross-attention. Each transformer uses self-attention within each modality (for its associated camera view) and cross-attention with the other modality (other camera view). As shown in Figure 7.2, we readout the CLS token from each transformer and concatenate them with the force-torque and proprioception embedding. This concatenated embedding is then processed using a 2-layer MLP policy head to output the robot actions.

Data Augmentations: Data augmentations have been shown to be helpful for single-task learning of manipulation tasks [117, 231]. However, naively using image augmentations can be detrimental for learning generalizable multi-task policies. This is because pixel-level augmentations, such as color-jitter, grayscale etc., can result in semantic changes in the overall scene. Such semantic changes can lead to mismatch between the input image and language instruction provided for the given task. For instance, a demonstration shows "move to red block" but pixel augmentations can change the red block's color. To avoid this while being able to utilize the benefits of augmentations we propose to use two different sets of augmentations. First, for third-person cameras we *only* use image-level augmentations (e.g. random crops, shifts). This avoids mismatch between image-and-text instructions and allows visual-language grounding from pretrained VLM to be utilized. Second, for first-person camera we use both image-level and pixel-level augmentations (color-jitter, grayscale). Since these augmentations lead to image-text mismatch this fur-

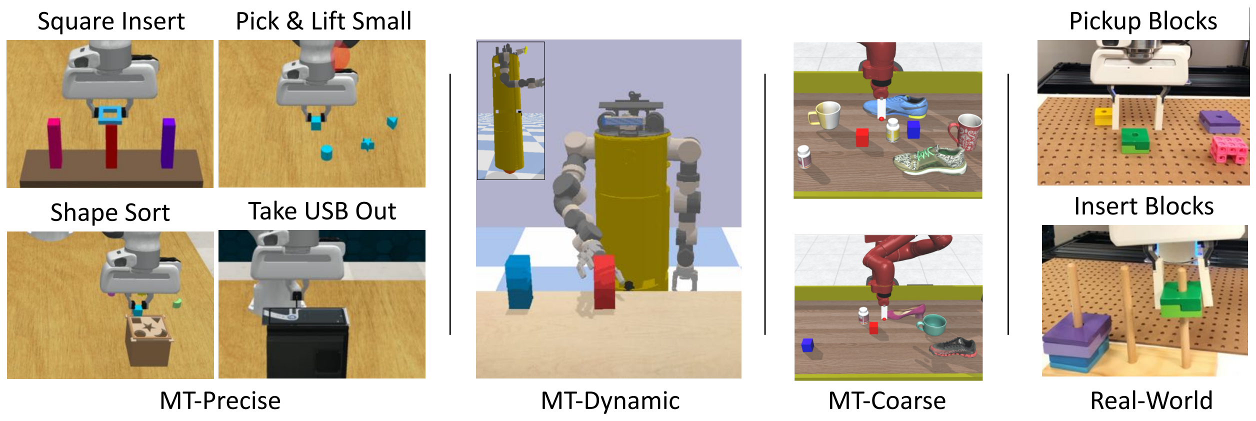


Figure 7.3: Task settings for evaluating our proposed approach. *Left*: Precision tasks. *Middle-left*: Dynamic tasks. *Middle-right*: Coarse tasks. *Right*: Real world pick and insertion tasks.

ther enforces our agent to use the third-person camera view for coarse localization, while only relying on the in-hand view for finer precise motions. Using strong pixel-level augmentations on first-person view further make the in-hand model invariant to texture but rely more on edges and corners [229]. This, as we show empirically, improves the generalization performance of our model on heldout object variations.

Training and Inference: We use behavior cloning from expert demonstrations to train our model. We record data from each sensor at their respective frequencies. Specifically, camera images are recorded at 30 Hz and force-torque feedback at 250Hz. To match slower processing times of larger models during inference we sub-sample the third-person camera images to 5Hz and first-person camera images to 20Hz. We use AdamW [101] optimizer with learning rate $1 \times e^{-4}$ and weight decay 0.01. We train our model for 60 epochs, using a linear warmup, starting with learning rate 0, for 5 epochs and then decay the learning rate using a cosine-scheduler. We use a GTX-1080Ti for inference. Overall our architecture has $\approx 250M$ parameters. The pretrained vision-language model has $\approx 150M$ parameters (for MDETR) with an inference time of ≈ 0.1 seconds. The first-person camera model has $\approx 25M$ parameters with an inference time of 0.04 seconds. Finally, the force-torque and proprioception model along with the policy head have a total of $\approx 250K$ parameters with an inference time of ≈ 0.005 seconds. This allows the actions to be inferred at a max frequency of ≈ 200Hz although we use it at a reduced frequency of ≈ 75Hz which was sufficient for our tasks.

7.4 Experimental Setup

We first identify the key research questions that we aim to evaluate:

Q1: How does **multi-spatial resolution** sensing benefit learning language-conditioned multi-task (MT) manipulation polices for precise tasks? Specifically, we aim to evaluate the utility of multi-spatial resolution sensing for tasks that involve visual occlusions, partial observability, and precision.

Q2: How does **multi-temporal resolution** sensor fusion benefit learning reactive manipulation tasks? Specifically, we evaluate how our architecture enables closed loop control for reactive tasks.

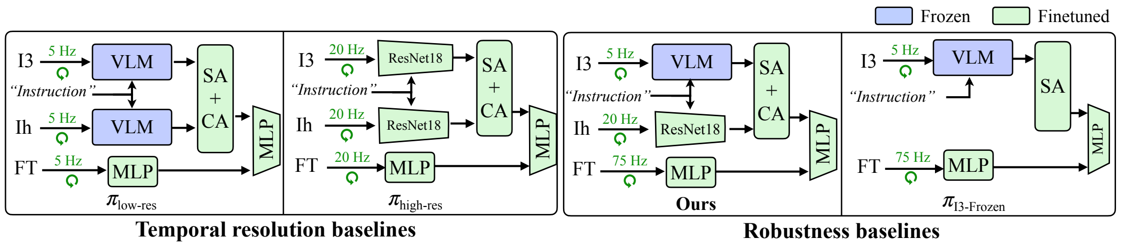


Figure 7.4: Temporal resolution and robustness baselines used to compare our multi-resolution approach.

Q3: How well does our approach **generalize** to tasks with novel visual-semantic targets? Specifically, we evaluate our approach's robustness to distribution shifts, e.g., object colors and geometries.

7.4.1 Environments

To evaluate the above questions we use three task settings, 1) **MT-Precise**: Precise manipulation tasks, 2) **MT-Dynamic:** Dynamic manipulation tasks, and 3) **MT-Coarse:** Coarse table-top manipulation tasks. Below we detail each environment and discuss its usage to answer above questions.

MT-Precise For precise manipulation we use 4 spatial precision tasks from RLBench [72] (see Figure 7.3 (Left)) – square block insertion, pick up small objects, shape sorting, and unplug usb. We use this task domain to answer **Q1**. Specifically, we evaluate the need for multi-spatial resolution sensing in manipulation tasks that require precise feedback and have partial observability, i.e., objects can go out of view of the first-person camera.

MT-Dynamic: We use the CMU ballbot [157] platform to perform dynamic pickup tasks in simulation (Figure 7.3 (Middle-Right)). We choose ballbot since it is a highly dynamic robot with an omnidirectional base (ball) capable of performing fast, reactive and interactive tasks. We consider the task of dynamically picking up an object, which requires quick reaction to contact with the object and grasping it to prevent toppling the object over. We use this setting to answer **Q2**.

MT-Coarse: We consider a canonical table-top manipulation setting ([258, 267]) involving coarse pick-and-place manipulation tasks with diverse objects – blocks, shoes, mugs, cylinders. We use this environment to answer **Q1** and **Q3**. Specifically, for **Q1** we contrast these coarse manipulation tasks with high precision tasks to evaluate the utility of multi-spatial resolution sensing.

Real-World Setup: We evaluate our approach on two real-world tasks. For precise manipulation (**Q1**) we use an *insertion* task to insert different blocks into cylindrical pegs (Figure 7.3 (Right top)). We also evaluate generalization abilities (**Q3**) using a *pickup* task, wherein we use 2 train objects and evaluate the learned policy on 8 objects with different geometry (shape, size) and visual (color, texture) features.

7.4.2 Baselines

We compare our approach against recent methods which focus on learning generalizable policies in multi-task settings. We compare against RT-1[20] which proposes a transformer based policy and also against

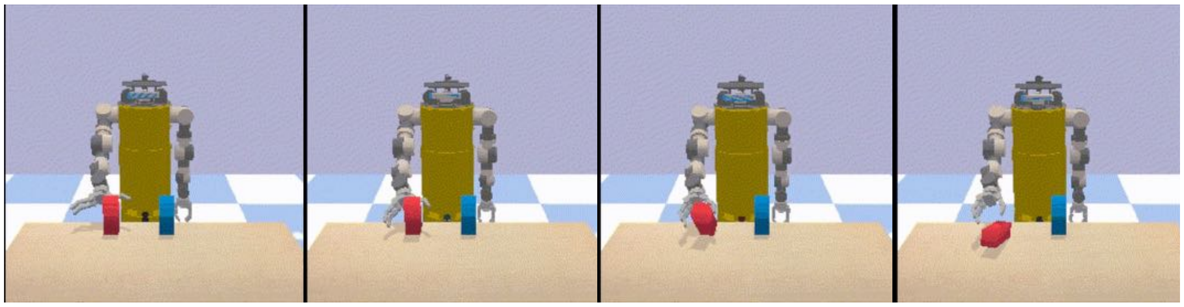


Figure 7.5: Example failure case for MT-Dynamic (Ballbot) task. As can be seen in the figure, if the robot approaches the object but does not react fast enough to the object contact, the block can topple resulting, in task failure.

BC-Zero [73] which uses language conditioning using FiLM [176]. However, both [20, 73] focus on coarse manipulation tasks and operate at a single-resolution (both temporal and spatial). To the best of our knowledge no prior work focuses on a multi-resolution approach for multi-task learning. Hence, to highlight the benefit of each component of our approach and answer the questions posed in Section 7.4 we modify our approach along different axes and propose additional baselines below.

Spatial Resolution baselines: To verify the utility of multiple spatial resolutions (**Q1**) we modify our approach and *remove* one sensory modality at a time. We use π_{-Ih}, π_{-I3}, π_{-FT} to refer to policies which *remove* first-person (**h**and view), **3**rd person view and **f**orce-**t**orque respectively.

Temporal Resolution baselines: To answer **Q2** we compare against single temporal-resolution approaches (Figure 7.4 (Left)), i.e., where all modalities (including force-torque) operate at the same frequency. We introduce two baselines, 1) $\pi_{\text{high-res}}$: small models with fast inference for both cameras (20Hz), and 2) $\pi_{\text{low-res}}$: larger models with slow inference for both cameras (5Hz).

Robustness baselines: We compare visual-semantic generalization ability of our approach (**Q3**) against two baselines (Figure 7.4 (Right)): 1) $\pi_{\text{multi-res-FT}}$: Finetune the pretrained VLM model, 2a) $\pi_{\text{I3-Frozen}}$: Uses only third-person camera (and force-torque) and keeps the pretrained model frozen. 2b) $\pi_{\text{I3-FT}}$: Uses only third-person camera (and force-torque) but finetunes the pretrained model.

Metrics: We use task success as the evaluation metric and report mean success over all tasks. During training, we evaluate the policy every 4 epochs and report average over *top-5* mean success rates across all evaluation epochs. For task generalization (**Q3**) we evaluate the train policy on novel visual-semantic tasks not seen during training. For all evaluations we use 20 rollouts per task.

7.5 Experimental Results

First, we evaluate the effectiveness of our multi-resolution approach against common multi-task baselines, RT-1[20] and BC-Zero[73]. We then present results for each research question. For qualitative results see: `https://sites.google.com/view/multi-res-real-time-control`.

	MT-Coarse	MT-Precise	MT-Dynamic
RT-1	81.0	12.5	4.5
BC-Z	74.1	7.8	4.8
Ours	82.0	55.0	73.6

Table 7.1: Task success comparison for multi-task baselines across all task domains.

7.5.1 Comparison to Multi-Task Baselines

Table 7.1 shows the results for the multi-task baselines RT-1[20] and BC-Zero[73] across all task We note that for coarse manipulation tasks (MT-Coarse) these baselines, that use single camera views, can perform quite well. This is because these tasks only require coarse localization of the target object for task completion. However, for precise manipulation tasks (MT-Precise), such baselines perform quite poorly since these tasks require fine-grained grasping (as many objects are $\approx$ 1cm in size) and insertion for successful task completion. domains. On the other hand, our multi-resolution approach, performs much better as it uses the first-person camera view and force-feedback for finer grasping and insertion. For dynamic tasks (MT-Dynamic), our method considerably outperforms the baselines (1.5x). This is because dynamic tasks require *reactive* response to contact events. Only our multi-temporal resolution approach utilizes high spatial and temporal resolution sensing, enabling fast response to contact events.

7.5.2 Additional Baseline Comparisons

Q1 – Spatial Resolution Experiments: We now compare against the *spatial* resolution baselines discussed in Section 7.4.2. For this set of baselines all methods use multi-*temporal* resolution sensing with high-frequency force-torque feedback. Table 7.2 shows results across all task settings. For MT-Coarse we see that only using a first-person camera (π_{-I3}) performs poorly. This is because of partial observability in this view, i.e., the target object can be out of view and lead to task failure. On the other hand, for MT-Precise (Row 2), only using first-person camera (π_{-I3}) performs better ($\approx 2\times$) than using only the third-person camera (π_{-Ih}). This is because MT-Precise tasks require finer motions which are hard to perform from low spatial resolution (third-person) view only. Further, for dynamic tasks (Row 3), using first-person views alone again suffers because of partial observability.

Q2 – Temporal Resolution Experiments: Table 7.3 compares against single-temporal resolution baselines ($\pi_{\text{low-res}}$ and $\pi_{\text{high-res}}$). Table 7.2 shows that for coarse and precise domains single-resolution perform as well as our multi-resolution approach. This is because tasks in both domains are quasi-static and hence fast reaction to contact events is not critical for task success. On the other hand, for dynamic tasks (Table 7.2 bottom row), since fast response to contact events is necessary (to avoid failures such as object toppling, see Figure 7.5 Appendix) our multi-resolution approach performs better than both $\pi_{\text{low-res}}$ (5Hz) and $\pi_{\text{high-res}}$ (20Hz) since it incorporates force feedback at 75Hz.

Q3 – Robustness Experiments: Table 7.4 compares results *(train / heldout)* for visual-semantic generalization against the robustness baselines in Section 7.4.2. As noted previously, for these experiments we evaluate the trained policies on *heldout* environments. We note that our approach, with frozen pretrained

	π_{-Ih}	π_{-I3}	π_{-FT}	Ours
MT-Coarse	74.5	41.0	81.8	82.0
MT-Precise	7.7	29.6	56.1	55.0
MT-Dynamic	65.8	27.5	33.2	73.6

Table 7.2: Results for multi-spatial resolution experiments (Section 7.5.2). Here, $-$ implies that we remove this input from policy. Thus, π_{-Ih} implies that the policy only operates on third-person camera views and force-torque feedback.

	$\pi_{\text{low-res}}$	$\pi_{\text{high-res}}$	Ours
MT-Coarse	82.0	81.0	82.0
MT-Precise	53.4	56.2	55.0
MT-Dynamic	4.2	12.2	73.6

Table 7.3: Results for multi-temporal resolution experiments (Section 7.5.2). Here, both $\pi_{\text{low-res}}$ and $\pi_{\text{high-res}}$ are single-resolution approaches which run at 5 Hz and 20 Hz respectively, while ours is a multi-resolution approach.

	$\pi_{\text{I3-Frozen}}$	$\pi_{\text{I3-FT}}$	$\pi_{\text{multi-res-FT}}$	Ours
MT-Coarse (Visual)	74.5 / 7.1	81.8 / 25.8	82.4 / 45.6	82.0 / 72.3
MT-Coarse (Geometry)	44.2 / 16.8	56.4 / 18.4	60.7 / 31.9	58.9 / 44.6
MT-Precise (Visual)	7.7 / 4.5	15.6 / 9.2	56.4 / 31.9	55.0 / 48.1

Table 7.4: Robustness experiment results, each cell shows *train*/*heldout* success rate (Section 7.5.2)

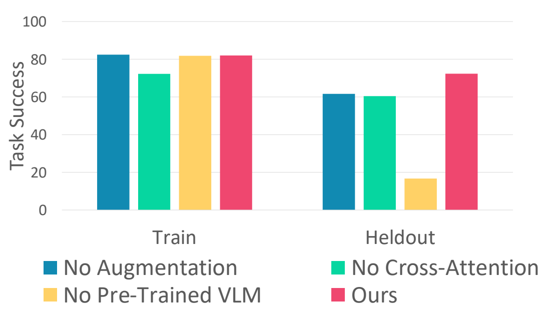
model, generalizes better than the finetuned model $\pi_{\text{multi-res-FT}}$. This shows the ability of our approach to maintain the generalization capabilities of the pretrained VLM as compared to the finetuned model that suffers from 'forgetting' and representation drift towards the training tasks. Additionally, from column-1 and column-2, we again note that the finetuned $\pi_{\text{I3-FT}}$ model suffers a larger decrease in performance as compared to $\pi_{\text{I3-Frozen}}$. Finally, comparing $\pi_{\text{I3-FT}}$ against $\pi_{\text{multi-res-FT}}$, we see that even with finetuning our multi-spatial resolution approach generalizes better because it can utilize first-person views for improved task success.

Real-World Experiments: We evaluate our approach in the real-world on two tasks, pickup and peg-insertion [263]. Table 7.5 shows comparison against the spatial resolution baselines. We note that our approach, with multi-spatial resolution, performs $\approx 3\times$ better than the baselines on both tasks.

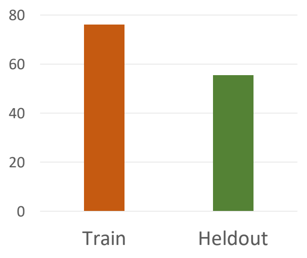
We see that given *limited* demonstrations both π_{-I3} and π_{-Ih} fail to perform well (across both tasks). On the other hand, removing force-torque feedback π_{-Ih} only affects performance on insertion task ($\approx 20\%$ less) since this task relies more on contact feedback. Additionally, Figure 7.6 (c) figure plots the robustness

	π_{-Ih}	π_{-I3}	π_{-FT}	Ours
Pickup	7.5 (3.5)	20.0 (14.1)	67.5 (3.5)	75.0 (7.0)
Peg-Insert	10.0 (0.0)	12.5 (4.6)	42.5 (3.5)	67.5 (3.5)

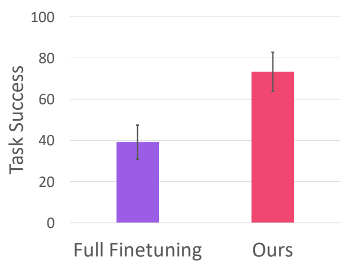
Table 7.5: Mean (stdev) results (using 2 seeds) for multi-spatial resolution for real world tasks.



(a) Ablation results



(b) Ablation results using pre-trained CLIP model



(c) Robustness results **(Q3)** for real-world Pickup

Figure 7.6: *Left:* Ablation results (see Section 7.5.3). *Right:* Robustness result for real-world pickup.

result for pickup task. As before we see that our approach with frozen model performs better. See website for qualitative results.

7.5.3 Ablations

For these set of results instead of using all 3 environment suites for evaluation, we choose the most appropriate environment suite for each component of our approach and evaluate on it.

Pixel-Level Augmentations: We evaluate the effect of pixel-level augmentations (color jitter, gray-scale) on the training and generalization of our MT-policies on MT-Coarse. Figure 7.6 reports results on both training and heldout (novel) evaluation configurations. We see that while there is very little difference in training performance, extensive pixel-level augmentations helps generalization by close to $\approx 15\%$. While pixel-level augmentations change the semantics of the task, our multi-modal approach is still able to complete the task because of visual-language grounded provided from pretraining.

Multi-Modal Fusion using Cross-Attention: We compare use of early fusion using cross-attention with late fusion using concatenation. Figure 7.6 shows that using cross-attention improves the performance by around $\approx 8\%$ on both train and heldout configuration. Thus, using cross-attention for multi-modal fusion is more effective than concatenation. However, we note that cross-attention requires more parameters and has slower inference.

Effect of Pretrained-VLMs: We also evaluate the effects of using pretrained-VLMs. Figure 7.6 shows the training and heldout performance using ImageNet initialization which only has visual pretraining and no vision-language pretraining. We see that while training performance matches our approach the heldout performance decreases tremendously. This large decrease is due to missing visual-language grounding since we use *separately* trained visual and language models.

Real-World Temporal-Resolution Comparison: We also ablate the effect of temporal resolutions on real-word robot performance. Specifically, we evaluate single temporal-resolution approaches ($\pi_{\text{low-res}}$) and $\pi_{\text{high-res}}$ for the peg-insertion task in the real-world. As before, to evaluate the learned policy we run each episode for a fixed duration of 60 seconds. However, we use early termination if the episode is solved successfully or the robot violates the desired workspace. Table 7.7 shows our results. Given that

Setup	BC-Z [73]	RT-1 [20]	Ours
Train	12.5	0.0	75.0
Eval	5.0	0.0	71.1

Table 7.6: Real-World results for using commonly used imitation learning (single-spatial resolution baselines) for Pickup task.

	$\pi_{\text{low-res}}$	$\pi_{\text{high-res}}$	Ours
RealWorld - PegInsert	45.0	62.5	67.5

Table 7.7: Additional Results for multi-temporal resolution experiments. As before, both $\pi_{\text{low-res}}$ and $\pi_{\text{high-res}}$ are single-resolution approaches which run at 5 Hz and 20 Hz respectively, while ours is a multi-resolution approach.

the insertion task is not dynamic, $\pi_{\text{high-res}}$ performs similarly to our approach. However, by comparison, ($\pi_{\text{low-res}}$) performs much more poorly (45% only). This is because a low-temporal resolution policy is not very reactive and hence doesn't respond fast to contacts made with the wooden peg. Thus, it is often unable to find the appropriate location to insert the block into the wooden peg. This can also be seen from qualitative videos (see success and failure videos), where both success and failure scenarios are much less reactive.

Temporal-Resolutions: Finally, we also ablate the temporal frequencies for the MT-Dynamic tasks. We ablate the effect of using camera inputs at low-resolution (third-person and in-hand camera inputs at 5Hz) while only force-torque feedback is used at high-resolution (75Hz).

Table 7.8 below shows our results. From the table below, we observe that the performance on MT-Dynamic tasks drops significantly when using the camera views at a very low temporal resolution. From our qualitative observations we note two common failure cases. First, where the ballbot is sometimes unable to reach the block to pick up. This is because, due to latency in the camera inputs (5 Hz), the policy outputs sub-optimal actions. Upon receiving updated camera inputs the policy tries to correct the trajectory. The overall resulting trajectory is noisy and fails to reach the target object. Second, again due to camera latency, the end effector does not align well with the target object and ends up toppling the object while trying to grasp it.

$\pi_{\text{low-res-high-FT}}$	Ours
33.4	73.6

Table 7.8: Results using low-temporal resolutions for camera-inputs (5Hz) and high-temporal resolutions for force-torque (75Hz).

7.6 Conclusion and Limitations

Our work proposes using sensing modalities at multiple spatial and temporal resolutions for learning multi-task manipulation policies. Our multi-resolution approach captures information at multiple hierarchies and allows the robot to perform both coarse and fine motions with high reactivity and real-time control. To learn generalizable multi-task policies we further leverage off-the-shelf pretrained vision-language models and freeze them to maintain their robustness. Our work has several limitations. While our proposed framework is general for multi-spatial sensing we only rely on global third-person camera and local first-person camera view. Further local sensing using vibro-tactile sensors [255, 260, 264] was not explored. Further, it is unclear if our approach of using cross-attention for sensor fusion will be optimal for more than 2 sensors. Additionally, while our multi-resolution policy allows us to learn robust policies not all sensing modalities will be available for all tasks. Thus, future work should explore adapting to scenarios with missing sensing modalities.

8 Conclusion and Future Work

8.1 Conclusion

Reusing knowledge learned from solving previous related or unrelated tasks to solve new tasks is a hallmark of human intelligence. In this thesis, we investigate different ways of reusing knowledge, in the form of representations, to perform new robot manipulation tasks. These representations were provided either as an inductive bias in the policy form or learned in a data-driven manner.

In Part-I of this thesis (Chapter 2 - Chapter 4) we focused on policy representations. As our policy representation, we proposed *parameterized* object-centric task-axes controllers (Chapter 2) which provide a geometric object-centric action abstraction for the robot agent. There are many advantages of this abstract action representation. First, it uses a compositional object-centric representation which makes it amenable to transfer to new tasks. Second, the geometric nature of task-axes controllers ensures that the learned policy is robust to many task-agnostic variations in the scene. Finally, these action representations allow the robot to learn robust policies for manipulation tasks efficiently in the real-world.

In the final chapter (Chapter 4) of Part-I we focus on using parameterized skill representations for lifelong learning of manipulation tasks. Our work relaxes prior works's assumptions on skill and task representations, enabling planning with more diverse skills and solving new tasks over time.

In Part-II of this thesis (Chapter 5 - Chapter 7) we focused on reusing visual representations for robot manipulation tasks. These visual representations are learned in simulation (Chapter-5) or learned from offline pretrained visual representations ((Chapter-6, Chapter 7). In Chapter 5 we *directly* use simulation trained 3D representations for learning skill preconditions. While Chapter-6 and Chapter 7 focus on using web trained visual representations. We show how directly using web-trained visual representations can be sub-optimal for moderately challenging manipulation tasks. Further, we show how adapting these representations via neural-adapters (Chapter-6) or by using additional sensory modalities (Chapter-7) can lead to much improved downstream robot manipulation performance.

8.2 Future Outlook

In this following sections, I briefly discuss some broad future ideas. While there are many ideas to pursue, I currently believe that scaling robot learning presents the most exciting and promising opportunities to improve real-world robots.

8.2.1 Benchmarking Policy Representations via Generative Simulations

In addition to reusing policy representations for learning new tasks we need to be able to share (deploy) our policy representations. However, sharing (or deploying) policies on robots that operate in the real-world is challenging due to safety considerations. To safely deploy our policy representations and reuse them in completely novel settings requires us to provide some guarantees or benchmarks on the performance of our policy representation. However, exhaustively benchmarking policies in many different real-world settings is challenging. This is because creating many diverse settings in real-world is infeasible. By contrast, physics simulators provide an efficient scalable alternative to this problem.

Physical simulators allow us to efficiently create novel scenarios with ease. Being able to efficiently generate many different scenarios allows us to benchmark our policy across many kinds of distribution shifts. However, simulations often bring their own set of challenges. First, simulations require appropriate 3D assets, which have always been a bottleneck. Uptil recently, large scale 3D asset datasets such as ShapeNet [28] have been severely limited. However, recent efforts such as CO3D [186] or Objaverse [38] provide a larger variety of 3D assets.

Besides large scale 3D assets, another issue with using simulators to benchmark policies is that creating simulations in the first place is still expensive. It requires considerable amount of user time as well as useful understanding of the underlying physics (e.g. contact dynamics, dynamics parameters). While this is insignificant for small scale simulations, this can be a bottleneck for creating large-scale simulation scenarios for benchmarking policies. However, given the impressive ability of large language models (LLMs) to generate code [191], utilizing LLMs to generate simulation scenarios is a plausible solution. Further, the common sense reasoning embedded in LLMs can be used to generate counterfactuals to robustly benchmark our policy representations.

8.2.2 Scaling Data for Robotics

Past few years have repeatedly demonstrated that data underlies the biggest wins within the wider machine learning community. Be it language, vision or robotics, large amounts of data has widely surpassed existing approaches in all of these fields. However, while language data is widely available on the internet and there is an increasingly large amount of visual data (e.g. YouTube), we do not have such data luxury in robotics. In fact, in robotics, we largely operate in data sparse settings.

Very recently, wide-scale community efforts [167] have focused on pooling large amounts of imitation learning data often collected in different academic and industrial labs. Such community efforts to crowd-source data will be crucial to scale robotics data. However, given that individual datasets are often collected in very limited lab settings, it is unclear if they provide sufficient diversity. One solution to this problem is to be able to provide *useful* robots to end-users. These end-users can take these robots home and collect data with them. However, such *useful* robots will need to have some basic skills and therein lies a chicken-and-egg problem. Recent efforts on building hardware for scaleable data collection and large scale robotics industry efforts (Tesla's Optimus, 1X, Figure), focus on similar problems and provide some glimmer of hope.

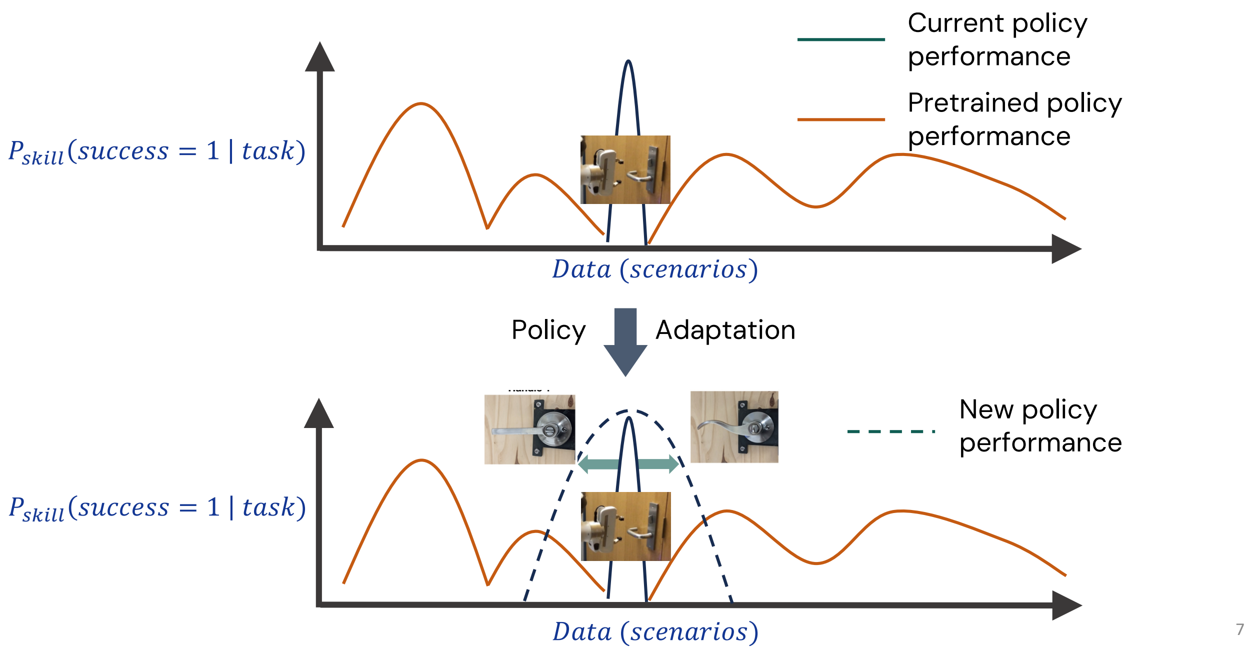

Current policy performance
Pretrained policy performance
$P_{skill}(success = 1 \mid task)$
Data (scenarios)
Policy
Adaptation
New policy performance
$P_{skill}(success = 1 \mid task)$
Data (scenarios)
7

Figure 8.1: How can we *robustly* adapt pretrained policies that have been finetuned with narrow data?

While real-world high quality data collection is important, alternative approaches to scale robotics data will become increasingly important. For instance, *real2sim* which uses scaleable simulations to collect increasingly large amounts of data. Alternatively, being able to utilize *video datasets* to bootstrap policy representations can provide useful avenues for future research. These approaches can focus on imputing the missing action labels in the video datasets, or extracting (building) structured policy representations, such as our task-axes controllers (Chapter 2), from them. These extracted structured policy representations can then be used to interact with the real world and *autonomously* collect larger datasets for further scaling.

8.2.3 Fast Real-World Adaptation

Fast adaptation (Figure 8.1) in the real-world is crucial for robotics. Reusing knowledge such as our task-axes controllers or using data-driven pretrained representations (Chapter 5 - Chapter 6) allow for fast adaptation in the real-world. However, each of these methods have their own set of limitations. For instance, [209] (Chapter 3) still requires few hours of trial and error learning. On the other hand, [212] (Chapter 5) requires collecting hundreds of data items (demonstrations, precondition learning data) for downstream tasks. Removing these limitations presents many opportunities for future research. As an example, learning from trial and error in the real-world, which often happens using reinforcement learning (RL) can be significantly improved. Data-driven priors such as proposing useful sub-goals, or which task-specific objects to interact with, or which controllers (low-level interaction policy) to use for certain sub-tasks can significantly reduce the exploration space of the robot agent. Alternatively, model-based approaches also provide an important and useful avenue for fast real-world adaptation [94].

Part IV

Appendices

9 Appendices

9.1 Appendix - Relational Learning for Skill Preconditions

9.1.1 Generating Pairwise Interactions in Simulation

To position the referrant object in the scene we sample a location (x, y, z) around the anchor object such that the distance between the anchor and referrant objects is less than 0.5 meter. We assume that the referrant object will be upright and hence to set its orientation we rotate it only around the Z-axis between $[-pi/6, pi/6]$ radians. To set the size of both the anchor and referrant objects we randomly sample each object dimension from $[0.04m, 0.20m]$. To save the voxel representation for each pairwise scene we set the voxel size to $0.01m$. To perform the local perturbation actions we use a virtual robot which can move an object around. Our virtual robot consists of prismatic joints which allow it to move along the XYZ axis and revolute joints to rotate around them. We use a spring-damper system to control these joints. For the prismatic joints we set $K = 100N/m$ and $C = 10Ns/m$, while for revolute joints we use $K = 10N/m$ and $C = 3Ns/m$.

9.1.2 Experimental Setup

Figure 9.1 shows our setup to collect real world data and perform skills based on the learned precondition models. We have 5 cameras mounted near the robot's workspace that allow us to create a full 3D scene. To create the 3D scene we collect point cloud data from all the cameras and project them onto a common frame. We finally use volumetric tsdf integration [34] to fuse all of the point clouds together.

Input Format. The input to our object relation mdoel and the baselines is of size $(C, 100, 100, 100)$, where C is the number of channels. We next detail the exact input format for both the baseline and object relation model.

Visual Baseline. Since the input scene contains variable number of objects we use two different input formats for our baseline methods. First, we only add masks for each object, i.e., each voxel is labeled with a value of 1 if it is occupied by any object, while unoccupied voxels are labeled with 0. Thus, this input format only requires one channel in its formulation. For our second input format we add another channel to the input. This additional channel contains labels for each object in the scene. Thus, if a voxel is occupied by $i'th$ object we label it with integer i. This input format consists of 2 channels. Additionally, we found using colored voxel representations to be extremely noisy since many objects in our experiments were quite close to each other.

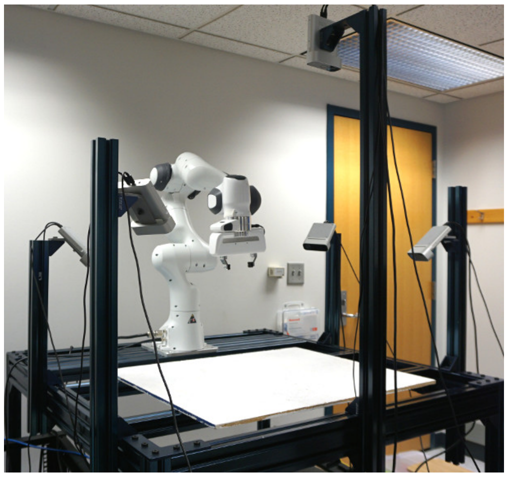


Figure 9.1: Experimental Setup with 5 cameras mounted around the robot arm. We combine the 3D point cloud output from each camera to create a full 3D scene.

We next discuss some changes to the input format for each specific task. For the food cutting task we add labels that represent the knife object and the food object if they are present in the scene. For all other objects in the scene we add a separate distractor label. For the block unstacking task we use two different labels. The block to be removed is marked by one label, while all the other objects in the scene use the other label. As before, we experiment with object ids for each block. But for the block unstacking task this method performed much worse and did not generalize at all when tested on a larger set of blocks.

Object Relation Model. The input to our object relational model f_{rel} consists of 3 channels, where the first channel contains binary masks for both anchor and referrant objects. The second and third channel contains the binary masks for the anchor and referrant object separately. When using the object relation model on real world manipulation data we take the object masks of two objects i and j. We set i as the anchor object and j as the referrant object. We transform the view such that the anchor object i is in the middle of the scene and centered at $(0, 0, 0)$. This is similar to the formulation used to train f_{rel} in simulation.

9.1.3 Real2Sim Baseline

As shown in the main paper we evaluate our approach against the real2sim baseline. The real2sim baseline we use the same voxel representations as used by our approach and other baseline methods. As the first step to implement the real2sim baseline we need to import the voxel representations from the real world to VREP [188]. Since these voxel representations do not conform to any fixed shape we cannot really import these object representations directly. Hence we import each voxel for every object separately into VREP. Thus each voxel is initially added as cuboid object. Once all the voxels of a given object are imported we group them together to form one composite object. Once all the objects in a scene have been imported we use it to test our preconditions. In addition to this, we also tried convex decomposition of the non-convex object shapes, but we found this to be incredibly slow and inaccurate for our purpose.

Since the VREP simulator does not support cutting we use the sim2real baseline only for the sweeping objects in a line and block unstacking task. To verify the preconditions for the sweeping objects in a line task we additionally add a virtual robot with a long cuboid attached to it. This robot consists of prismatic joints which allow us to move in the XYZ axes. As before we use a spring-damper system to move the robot. At the beginning of the task we move the robot to its start position. We set the initial X coordinate as the minimum X-position of all object corners minus some threshold $(0.02m)$. The Y and Z coordinates are set as the median of all object centers. This allows us to sweep through the scene in a consistent manner. To verify if the objects are indeed along a line, we evaluate if all the objects lie on one side (along the X-axes) of the robot handle.

We verify the preconditions for the block unstacking task by verifying if the blocks in the scene minus the block to be removed are stable or not. To achieve this we import all the blocks except the one to be removed into the simulator. Initially, all blocks are kept static. Once all of them have been imported we convert them to dynamic and allow them to interact with each other. To verify if the blocks are indeed stable we note each block's position before and after allowing dynamic interactions. If any of the blocks move by more than a certain specified threshold we assume that the blocks are not stable. To find the best threshold value we do a grid search in the set $(0.0005, 0.001, 0.0015, 0.002, 0.003, 0.004)m$. We found 0.0015 and 0.002 to give the best and very similar results.

9.1.4 Architecture Details

Baselines. For our baseline architecture we adapt the resnet model for 3D input [59]. However, we slightly tweak the ResNet architectures in [59] for our purposes. Additionally, different from [59] for the first convolution layer we use the same stride for all three input channels. Besides these changes, we follow the same architecture as [59]. For other specific architecture details please look at Table 1 in [59]. The output of the ResNet model is forwarded through 2 fully-connected layers with outputs of size 64 and 1 respectively.

We also use another baseline similar in style to VGG [223], *i.e.*, our model consists of a series of convolutions and ReLU non-linearities. More-specifically our architecture consists of 6 convolution layers, each of which operates on 3D input. Table 9.2 lists the different parameters for each of these convolution layers. The output of each convolution layer is passed through ReLU non-linearity. The final output of the convolution is flattened to a 512 dimensional vector. This vector is then passed through multiple fully-connected layers with output size $[256, 64, 1]$.

Precondition Learning Model. We next detail the precondition learning model's for both architecture choices, i.e., using a relational network or a graph neural network.

Relational Network. For our relational network model we use the below function,

$$RN(S) = g_\phi\left(\sum_{i,j} f_\theta(f_{\text{rel}}(o_i, o_j) \oplus f_{\text{rel}}(o_j, o_i))\right), \tag{9.1}$$

where $\oplus$ represents the concat operation, the function f_θ consists of two fully-connected (FC) layers with ReLU non-linearity in-between, and g_ϕ also consists of 2 FC layers with ReLU non-linearity.

We implement the relation network model f_θ using a two layer neural network with fully connected layers with outputs of size 128 and 32 respectively. We implement g_ϕ, which acts as an accumulator, again by using two fully connected layers with outputs of size 16 and 1 respectively. We use the ReLU non-linearity for both f_θ and g_ϕ.

Graph Neural Network: For our GNN model we use the following update model to process the output at each node,

$$v_i^{(1)} = f_\psi\left(v_i \oplus \sum_j \Big(f_{\text{edge}}(v_i^{(0)}, v_j^{(0)}, e_{ij}) \Big) ; \psi \right) \tag{9.2}$$

We add separate node and edge models for our GNN model. Our edge model takes the edge input and passes it through a fully connected layer with output size of 128, followed by ReLU, and finally with another fully connected layer which outputs a continuous embedding for this edge. These edge embeddings are used for the next graph layer as well as for each node being processed. Our node model initially sums the edge embeddings for all the edges connected to this node. This sum of edge embeddings is then concatenated with the original node input and passed through a fully-connected layer with output size 128, followed by ReLU and then another fully conected layer with output size 128. This is similar in design to the edge update model. We stack two layers of the above GNN model for precondition learning. The first layer uses an input of size 256 and outputs representations of size 128. The next layer uses an input of size 128 and outputs representations of size 128. We sum the node and edge embeddings of the last GNN model and concatenate them together to form an input of size 256. This input is then passed through 2 fully conected layers with output size of 64 and 1 respectively.

Object Relation Model

Our object relation model initially uses the ResNet-18 model of [59]. As before, we make two changes to this model, we use a stride of $(2, 2, 2)$ for the first convolutional layer, and we do not use the BatchNorm layers. The convolutional layer output is flattened into a vector of size 1536 which is then projected through 2 fully-connected layers with output size 512 and 256. Thus, we get the final relational embedding of size 256. We concatenate this with the action vector and pass them through three fully connected layers of size 128, 64 and 9 respectively. The final output of size 9 consists of the predicted position changes of size 3, predicted orientation changes of size 3 and predicted mean contact position of size 3.

9.1.5 Training Details

Table 9.1 lists some of the training details for the different models used in our approach. Below we describe training details for each model separately.

Object Relation Model. As noted previously, for our embedding network we use a 3D CNN with a ResNet [63] based architecture [59], specifically we use the ResNet-18 architecture. The output of the ResNet18 model is then downsampled using a linear layer to a size of 256. To train the above embedding model, we use the Adam optimizer [101] and set an initial learning rate of $3e-4$ and a batch size of 256. Our overall loss function for the

Model	Batch Size	Learning Rates
Object Relation	256	3e-4
RN based Precondition	8	{1e-3, 3e-4}
GNN based Precondition	8	{1e-3, 3e-4}
Baseline (Resnet-18/34)	4, 8, 16	[1e-3, 1e-4]
VGG	4, 8, 16	[1e-3, 1e-4]

Table 9.1: Batch Size and Learning rates for different models used

	conv-{1,2}	conv-{3, 4, 5, 6}
kernel	5	3
padding	(2, 2, 2)	(2, 2, 2)
stride	(1, 1, 1)	(1, 1, 1)

Table 9.2: Parameters for the convolution layers of our VGG* network.

Parameter	Value
$\Delta p^r_{\text{sim}}, \Delta p^r_{\text{diff}}$	0.2, 0.21
$\Delta\theta_{\text{sim}}, \Delta\theta_{\text{diff}}$	0.004, 0.008
γ for Δp	2.0
γ for $\Delta\theta$	2.0

Table 9.3: Different hyper-parameters for contrastive loss formulation.

The total loss for the object relation model can be written as,

$$\begin{aligned}\mathcal{L} &= \lambda^{\text{cont}}_{\text{pos}} \times \mathcal{L}^{\text{cont}}_{\text{pos}} + \lambda^{\text{cont}}_{\text{orient}} \times \mathcal{L}^{\text{cont}}_{\text{orient}} \\ &\quad + \lambda_{\text{pred}} \times \mathcal{L}_{\text{pred}} + \lambda_{\text{orient}} \times \mathcal{L}_{\text{orient}} + \lambda_{\text{contact}} \times \mathcal{L}_{\text{contact}},\end{aligned}$$

where the first set of losses $\mathcal{L}^{\text{cont}}_{\text{pos}}$, $\mathcal{L}^{\text{cont}}_{\text{orient}}$ are the position and orientation based contrastive losses. The next set of losses $\mathcal{L}_{\text{pred}}, \mathcal{L}_{\text{orient}}, \mathcal{L}_{\text{contact}}$ are the direct supervised losses. When training the object relation model with all the above losses we set, $\lambda^{\text{cont}}_{\text{pos}} = 2$, $\lambda^{\text{cont}}_{\text{orient}} = 2$. For the supervised losses we set $\lambda_{\text{pos}} = 1$, since orientation values change less we set $\lambda_{\text{orient}} = 10$, and finally $\lambda_{\text{contact}} = 1$. Alternatively, when training the model with just contrastive losses we set $\lambda^{\text{cont}}_{\text{pos}} = \lambda^{\text{cont}}_{\text{orient}} = 1$.

Contrastive Loss: To implement the contrastive loss we use a batch all strategy, *i.e.*, instead of explicitly sampling anchor, positive and negative pairs separately, we sample a batch of data and compare all scene triplets. We compare normal and adaptive actions for each triplet pair separately, *i.e.*, for each scene in the triplet we compare their adaptive action effects with the other scene's adaptive action effects only. To compare normal actions with different action magnitudes we use a threshold of $0.04m$ to classify similar actions. Thus, scenes with difference in action magnitude greater than the above threshold are not compared directly in contrastive losses.

As noted in the paper, we use the following to compare the action effects on position changes,

$$\Delta p^r = \frac{\Delta p^{\text{observed}}}{\Delta p^{\text{desired}}}, \tag{9.3}$$

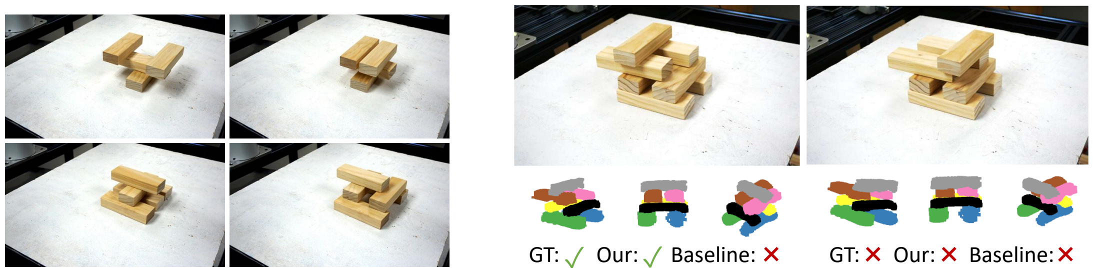


Figure 9.2: *Left:*: Example train scenes with 4 and 5 blocks used in the block unstacking task. *Right:* Sample test scenes which are correctly classified by our model but incorrectly by the learning based baselines. In both images the top block is at different locations.

Model	Train Set	Test Set	F1	Wt-F1
ResNet-18	0, 1, 2 distractors	3 distractors	0.844	0.902
ResNet-34	0, 1, 2	3	0.877	0.871
VGG*	0, 1, 2	3	0.788	0.842
Real2Sim	-	-	N/A	N/A
Our Model (RN)	0, 1, 2	3	0.880	0.935
Our Model (GNN)	0, 1, 2	3	**0.921**	**0.940**

Table 9.4: Precondition learning results for cutting food skill.

where $\Delta p^{\text{observed}}$ is the observed change in referrant object center position, and $\Delta p^{\text{desired}}$ is the desired change i.e. action vector. We set $\Delta p^{\text{desired}}$ to 1 for normal actions, while for adaptive actions it is set to the voxel distance between objects centers. To find scenes with similar action effects we use a threshold of 0.2. More precisely, given two scenes (m and n) and their action effects in terms of Δp^r, we say these two scenes are similar if their action effects are below the threshold for all actions, $|\Delta p^r_m - \Delta p^r_n| < 0.2$. Analogously, if $|\Delta p^r_m - \Delta p^r_n| > 0.21$ we set these two scenes as dissimilar. For orientation changes we set $\Delta\theta_{\text{sim}} = 0.004$ and $\Delta\theta_{\text{diff}} = 0.008$. Finally, we set the contrastive loss margin $\gamma = 2.0$ for both position and orientation changes. Table 9.3 lists the above parameters for contrastive losses.

Precondition Learning Models. Our Model: For our RN and GNN based models, we use the Adam optimizer [101] and set an initial learning rate of $3e-4$. For accelerated learning, we also test with a larger learning rate of $1e-3$. Since the precondition model is trained on less data we use a smaller batch of size 8 only. We also decay the learning rate with 0.995 after every epoch.

Baselines: For the baseline models we use the Adam optimizer as well. We experiment with both random initialization as well as kaiming initialization for the resnet based baseline methods [63]. We do a grid search to find the best learning rate between $[1e-3, 1e-4]$ and report numbers with it. Also, given the small train data size we test with multiple batch sizes of 4, 8 and 16 and report numbers with the best results.

Table 9.1 lists the different parameters used to train the precondition models.

Model	Train Set	Test Set	F1	Wt-F1
ResNet-18	6, 7	4	0.735	0.765
ResNet-34	6, 7	4	0.765	0.784
VGG*	6, 7	4	0.688	0.712
Real2Sim	-	4	**0.89**	**0.86**
Our Model (RN)	6, 7	4	0.74	0.764
Our Model (GNN) all edges	6, 7	4	0.772	0.798
Our Model (GNN) sparse edges	6, 7	4	0.824	0.825

Table 9.5: Results for precondition learning of block unstacking task with 6 and 7 objects in the train set and 4 objects in the test set.

9.1.6 Task Setup

Cutting Food: For the food cutting data we use a custom 3D printed tool holder with an embedded knife that the robot can grasp and use to cut food items. We use multiple different target food items and obstacles with different shapes and sizes. Unlike the sweeping objects in a line task we also add knife and food labels to their respective voxel representations, while all the other objects have no label. We create scenes with food, knife and multiple objects with a maximum of upto 6 objects in the scene.

Block Unstacking: To create the dataset we pre-program the robot to assemble a block of stacks in different configurations and then manually label which blocks are crucial for stability. We create scenes with variable number of blocks from 3 to 7. For each set of blocks we create between 10 to 20 scenes with different configurations. In addition to the previous models, we add another GNN based model where graph edges only exist between blocks (vertices) that are closer than a certain threshold (0.1m). This allows us to remove edges between blocks which are far apart and thus eases the learning problem.

9.1.7 Additional Results

In this section we discuss some additional results for our main experiments.

Cutting Food. Table 9.4 shows the results for the food cutting experiment when we have 3 distractor objects in the test set, but only upto 2 distractor objects in the train set. As observed above, the GNN based model performs the best with a wt. F1-score of 0.94. Additionally, the baseline models also perform well with a maximum wt. F1-score of 0.902. The good performance of the baseline models on food cutting task can be attributed to the fact that there exist only two main objects in the scene (food and the knife). Since we provide separate labels to both of these objects, the baseline models only needs to focus on these objects and hence does not require complex compositional reasoning. Thus, even with increasing number of distractor objects, the baseline model still performs quite well.

Block Unstacking. For the block unstacking task we also experiment with the scenario where the train set contains larger number of objects as compared to the test set. Table 9.5 shows results when the train set contains 6 or 7 objects while the test set contains only 4 objects. Interestingly, the performance of our GNN based models also reduces in this scenario. We get a max. wt. F1-score of 0.825 only, which is worse as compared to the previous train-test splits. However, our GNN based models still perform better than the visual baseline models, whose maximum wt F1 score is 0.784. This result indicates that although structured representations are extremely useful, we cannot assume that training on a large set of objects will automatically generalize to fewer objects as well. The real2sim baseline performs the best (wt. F1 score: 0.889) in this setting. This is because using 4 blocks, we can only create a limited number and types of block stacks. Hence, given only stacks with 3 blocks (since we remove 1 block before exporting into the simulator) the Real2Sim baseline is able to perform quite well. Additionally, a small number of blocks also result in fewer perception errors, these errors when transferred to simulation do not affect the precondition output significantly. This is also similar to the real2sim results for sweeping task (main paper) wherein we observe that larger number of objects lead to worse performance.

Figure 9.2 (Right) shows example scenes which show the complex reasoning required for solving the block unstacking task. The only difference between the two scenes in Figure 9.2 (Right) is in the position of the top block which results in different ground truth precondition labels. While our precondition model is correctly able to predict the preconditions in both scenes, the visual baseline model always predicts false. The above example clearly shows that our model is able to perform complex reasoning using the learned object relational embeddings.

9.2 Appendix - Hiearchical Object-Centric Controllers for Robotic Manipulation

9.2.1 Controller Implementation Details

Specific Controllers for each Task. Below we list out the controllers used for each task in our experimental setup.

Block Fit. A set of controllers is associated with each wall in the environment. For a wall, let v be the unit vector pointing in the wall's normal direction, x_{wm} be the coordinate of the middle of the wall. The set of controllers associated for each wall include:

1. Position attractor along normal direction. $x_d = x_{wm}$, $u = v$
2. Position attractor along error direction. $x_d = x_{wm}$, $u = \frac{x_d - x_c}{\|x_d - x_c\|_2}$
3. Force attractor along the normal direction. $f_d = 10$, $u = v$
4. Rotation attractor aligning the block's x-axis to the normal. $r_d = v$, $u = [1, 0]^\top$
5. Rotation attractor aligning the block's y-axis to the normal. $r_d = v$, $u = [0, 1]^\top$

Block Push. In addition to all the per-wall controllers of the Block Fit task, Block Push has the following per-wall controllers:

1. Position controller along the side of a wall. Let v' be a unit vector orthogonal to v. Since there are 2 such possible directions, we pick the one that gives the direction pointing up along the vertical wall in the scene.

   Let x_{wc} be the coordinate of a wall corner. Since walls form a corner-connect chain in this task, using one of the two corners per wall covers all corners in the scene except the last corner in the chain, which we ignore.

   With these, this controller has $x_d = x_{wc}$ and $u = v'$.

2. Position curl controller around a wall corner. This controller rotates the end-effector in a fixed-radius circle around a point until it reaches the target position which also lies on the circle. The attractor target is $x_d = x_{wc} + \|x_c - x_{wc}\|_2 v'$, and the direction is $u = R(\frac{\pi}{2}) \frac{x_c - x_{wc}}{\|x_c - x_{wc}\|_2}$, where $R(\theta)$ gives a 2D rotation matrix with the angle θ.

Block Push has one more position controller that attracts the robot block toward the target block. Let x_g be the current location of the center of the target block. This position controller has $x_d = x_g$ and $u = \frac{x_d - x_c}{\|x_d - x_c\|_2}$.

Franka Hex-Screw. Let x_s be the location of screw, and $x_g = x_s + [0, 0.02, 0]^\top$ be a point 2cm above the screw (the y-axis is vertical in our coordinate frame). Position attractor controllers use x_g as the tar-

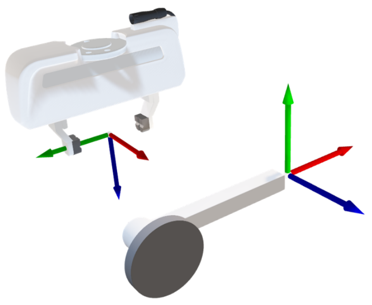


Figure 9.3: Axes Visualization for Franka End-Effector and Door Handle for the Door-Open Task. RGB corresponds to XYZ.

get, instead of x_s, because attracting the hex-key tip toward the inside of the screw directly can result in collisions with the side of the screw and prevent the key from properly inserted.

1. Position attractor along vertical direction. $x_d = x_g, u = [0, 1, 0]^\top$
2. Position attractor along error direction. $x_d = x_g, u = \frac{x_d - x_c}{\|x_d - x_c\|_2}$
3. Position controller that prevents motion in the vertical direction $x_d = x_c, u = [0, 1, 0]^\top$. This controller does not attract the end-effector toward a goal. Instead, its utility is solely in its nullspace projection, which ensures lower-priority controllers cannot move the end-effector outside of a horizontal plane. This controller is useful for preventing prematurely inserting the hex-key.
4. Force controller that pushes downward toward the hex screw. $f_d = 20, u = [0, -1, 0]^\top$.
5. Rotation controller that maintains the verticality of the end-effector. $r_d = [0, 1, 0]^\top, u = [0, 0, 1]^\top$. The positive z-axis of the end-effector frame corresponds to the direction that the hex-key points towards.
6. Rotation controller that rotates the hex-key counter-clockwise. $r_d = R_y(100°)[1, 0, 0]^\top, u = [1, 0, 0]^\top$, where $R_y(\theta)$ gives a rotation matrix that rotates around the y-axis with the angle θ.
7. Rotation controller that rotates the hex-key clockwise. $r_d = R_y(-100°)[1, 0, 0]^\top, u = [1, 0, 0]^\top$

Franka Door-Open. See Figure 9.3 for a visualization of both the Franka end-effector and door handle axes. Let $r_{\{x,y,z\}}$ correspond to the 3 axes of the door handle. Let x_g be a grasp point on the door handle, x_h be the center of the handle axle (dark gray cylinder in Figure 9.3). The set of controllers include:

1. Position attractor to door handle along error direction. $x_d = x_g, u = \frac{x_d - x_c}{\|x_d - x_c\|_2}$
2. Position curl attractor for rotating around the handle in the plane of the door panel (the nullspace of r_z). Let $x_e = \mathcal{N}(r_z)(x_c - x_h)$. Then $x_d = x_h - \|x_e\|_2 r_y, u = \frac{x_e}{\|x_e\|_2} \times r_z$
3. Force controller to pull the handle. $f_d = 50, u = -r_z$

4. Rotation controller to align the x-axes of the gripper and the handle. $r_d = r_x, u = [1, 0, 0]^\top$
5. Rotation controller to align the y-axes of the gripper and the handle. $r_d = r_y, u = [0, 1, 0]^\top$
6. Rotation controller to align the z-axes of the gripper and the handle. $r_d = r_z, u = [0, 0, 1]^\top$

Integral Term for Force Controllers. Using an integral term for force controllers can help reduce the force error and improve stability. Let $\bar{\delta_f^i}$ be the accumulated force errors for the force controller used at the ith priority. Then, the corresponding delta position target is computed as:

$$\Delta_x^i = \mathcal{N}_i(K_f \delta_f(f_d^i, u^i, f_c) + K_I \bar{\delta_f^i}) \tag{9.4}$$

where $\mathcal{N}_i = \mathcal{N}([u_0, \ldots, u_{i-1}])$.

Delta Target Magnitude Clipping. To ensure safety and limit the maximum speed at which our controllers can drive the robot, we clip the magnitude of delta position and rotation targets.

Let D_x be the maximum delta translation magnitude corresponding to a position controller, and D_f for a force controller. The clipping for force and position controllers are computed as follows:

$$\Delta_x^i \leftarrow \frac{\min(\|\Delta_x^i\|_2, D_*)}{\|\Delta_x^i\|_2} \Delta_x^i \tag{9.5}$$

Note that D_* can be D_x or D_f, depending on if the ith controller is a position or force controller.

Similarly, let D_R be the maximum delta rotation angle for rotation controllers:

$$\Delta_R^i \leftarrow \frac{\min(\|\Delta_R^i\|_2, D_R)}{\|\Delta_R^i\|_2} \Delta_R^i \tag{9.6}$$

Controller Hyperparameters. Table 9.6 lists the different hyperparameters used for the object axes-controllers for each task. We list the gains used for each controller as well as the clipping used while executing each controller. Table 9.7 lists the task-space impedance parameters used for simulation and real-world experiments. We use [263] to implement each controller for real-world experiments.

	Block Fit	Block Push	Franka Hex-Screw	Franka Door-Open
D_x (m)	1	0.5	0.03	0.03
D_f (N)	0.5	0.1	10^{-4}	10^{-4}
D_R (deg)	90	120	10	10
K_x	1	1	1	1
K_f	1	1	1	1
K_I	0	0	10^{-4}	10^{-4}

Table 9.6: Controller Gains and Magnitude Clips Across Tasks.

	Simulation	Real World
K_S	1000	600
K_D	$2\sqrt{1000}$	$2\sqrt{600}$
T	10	30

Table 9.7: Task-Space Impedance Control Parameters. K_S is stiffness, K_D is damping, and T is how many timesteps a controller combination runs before the RL policy is queried again. The simulation and real-world values are not the same due to differences in control frequencies and Franka dynamics between real-world and simulation. We tune the real-world values to ensure that the resultant controller behaviors are similar to those in simulation. This tuning was done prior to task evaluations.

9.2.2 Task Details

Block Fit *Observations.*

1. 2D pose of block robot
2. 2D contact force direction and magnitude experienced by the block robot in the world frame.
3. 2D coordinates of centers and wall corners

Reward Function: Let ϕ_d be the previous distance between the block translation and the goal translation, ϕ_θ be previous the absolute angle difference between the block rotation and the goal rotation, and let ϕ'_d, ϕ'_θ be there current counterparts. The reward function rewards making progress towards the goal with a small alive penalty and a large task completion bonus:

$$R = 10(\phi'_d - \phi_d) + 5(\phi'_\theta - \phi_\theta) - 0.1 + 1000 \times 1(\phi'_d < 0.16 \wedge \phi'_\theta < 5^\circ) \tag{9.7}$$

The goal translation threshold 0.16 is about half the width of the block.

Block Push

Observations.

1. 2D pose of block robot
2. 2D contact force direction and magnitude experienced by the block robot in the world frame.
3. 2D coordinates of centers and wall corners
4. 2D pose of the target block

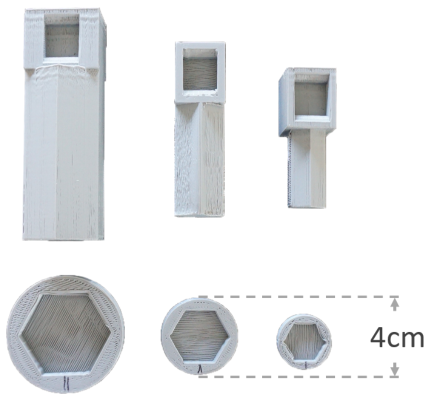


Figure 9.4: Different hex screw and key sizes used for testing in the real world. The middle size represents $1.0\times$ scale factor, while the left is $1.5\times$, and the right $0.7\times$.

Reward Function: The reward function is similar to that of Block Fit, except the progress rewards are computed w.r.t. the target block, not the block robot, and there is an additional reward term for approaching the target block:

$$R = 10(\phi_d' - \phi_d) + 10(\phi_b' - \phi_b) - 0.1 + 200 \times 1(\phi_d' < 0.05) \tag{9.8}$$

where ϕ_b is the previous distance between the robot block and the target block, and ϕ_b' is the current counterpart. The goal translation threshold of 0.05 is about half the width of the target block.

9.2.3 Franka Hex Screw

See Figure 9.4 for the different screw and key sizes used in real-world experiments.

Observations.

1. 7-dimension robot arm joint angles
2. Gripper width
3. 6D pose of the tip of the hex-screw. Rotations are represented via quaternions
4. End-effector contact forces
5. Position of the hex screw relative to the robot base

Reward Function: Let ϕ_d be the previous distance from the hex-key tip to the hex screw, ϕ_θ be the previous absolute angle difference between the screw angle and its target angle, and let ϕ_d', ϕ_θ' be their respective current counterparts. Let ρ be the absolute angle difference between the negative y-axis (pointing downwards) and the z-axis of the end-effector. The reward function rewards approaching the hex-screw, making progress in turning the screw, maintaining a vertical hex key orientation, plus a small alive penalty and a large task bonus:

$$R = 400(\phi'_d - \phi_d) + 10(\phi'_\theta - \phi_\theta) - \rho - 1 + 1000 \times 1(\phi'_\theta < 5^\circ) \tag{9.9}$$

The target screw rotation angle (at which point $\phi_\theta = 0$) is 70°.

Franka Door Opening

Observations.

1. 7-dimension robot arm joint angles
2. Gripper width
3. 6D pose of the tip of the hex-screw. Rotations are represented via quaternions
4. End-effector contact forces
5. Door panel angle (How much the door has opened, not the angle of the door handle)
6. Position of the door handle relative to the robot base

Reward Function: Let ϕ_d be the previous distance from the end-effector to the door handle grasp point, ϕ_θ be the previous absolute angle difference between the door handle angle and the target handle turning angle, ϕ_ρ be the previous absolute angle difference between the door panel angle and the target door opening angle, and let ϕ'_d, ϕ'_θ, ϕ'_ρ be their respective current counterparts. Let c denote the current end-effector contact forces. The reward function rewards approaching the door handle, turning the handle, turning the door, plus small alive penalties and excessive force penalties, plus a large task bonus:

$$R = 10(\phi'_d - \phi_d) + 10(\phi'_\theta - \phi_\theta) + 100(\phi'_\rho - \phi_\rho) - 0.01 - 0.001\|c\|_2 + 100 \times 1(\phi'_\rho < 5^\circ) \tag{9.10}$$

The target door handle turning angle (at which point $\phi_\theta = 0$) is 90°, and the target door panel opening angle (at which point $\phi_\rho = 0$) is 35°.

9.2.4 RL Training Details

PPO Hyperparameters

Table 9.8 lists the hyperparameters used for each of the experiments. In addition to the above parameters, we also decay the clip range using a linear schedule with a decay rate of 0.99 after every epoch. We set the minimum clip range value to be 0.04. Also, for the Franka Hex-Screw and Franka Door-Open task we evaluated a range of entropy coefficient values $[0.01, 0.1]$ for the end-effector action space.

Network Architecture. For all tasks and methods, we use the same network architectures for both the policy and value function networks. The network consists of 2 hidden layers with 256 hidden units each.

	Block Fit	Block Push	Franka Hex-Screw	Franka Door-Open
num steps	500	240	450	480
discount factor	0.995	0.995	0.995	0.995
entropy coefficient	0.01	0.01	$[0.01, 0.1]$	$[0.01, 0.1]$
learning rate	2.5×10^{-4}	2.5×10^{-4}	2.5×10^{-4}	2.5×10^{-4}
value loss coefficient	0.5	0.1	0.5	0.5
max gradient norm	0.5	0.5	0.5	0.5
lambda	0.95	0.95	0.95	0.95
num minibatches	50	30	30	50
num opt epochs	4	4	4	4
clip range	0.2	0.2	0.2	0.2

Table 9.8: PPO Hyperparameters Across All Tasks.

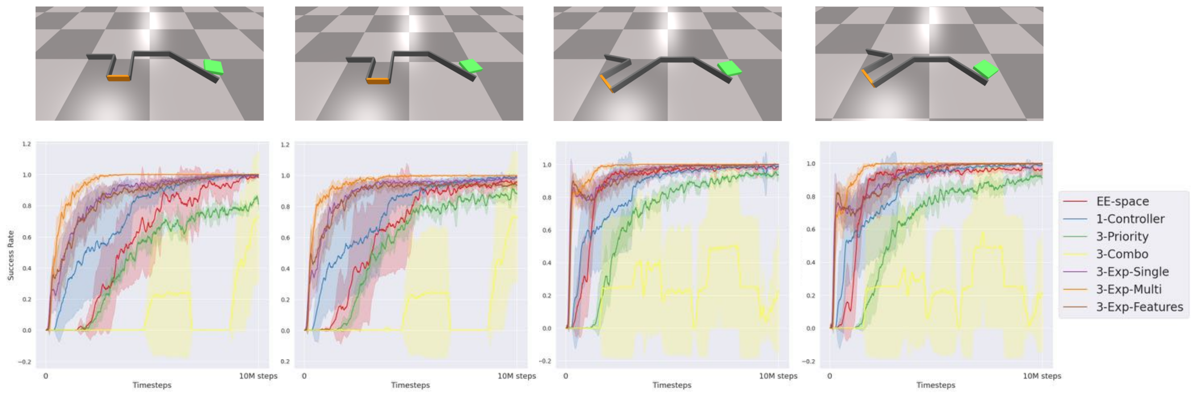
9.2.5 Detailed Experiment Results

We discuss results for each task in more detail in the following sections. Video results for all the different tasks and methods can be seen at `https://sites.google.com/view/compositional-object-control/`.

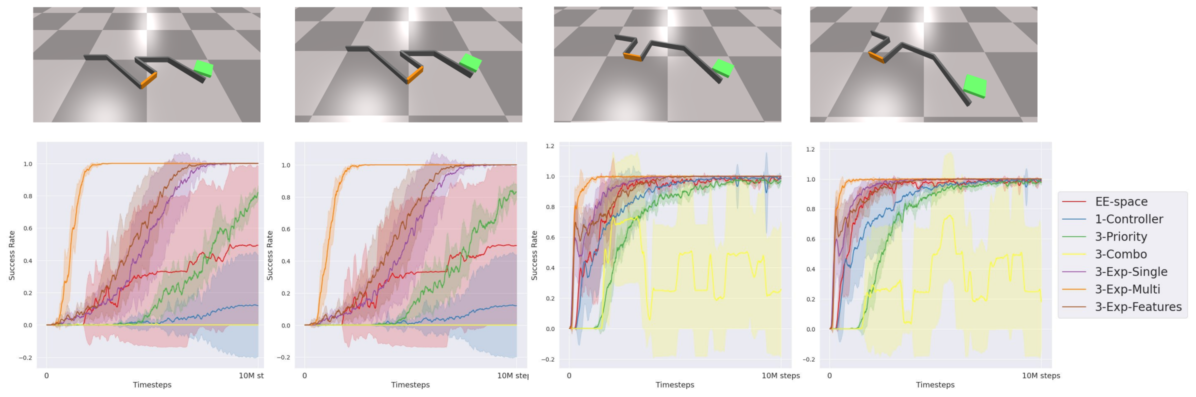
Block Fit

Figure 9.5 plots the train results (success-rate) for all different train configurations. As can be seen in the above figure, for all configurations besides a couple (Figure 9.5b left two plots), all the methods perform quite well during training. The poor performance in two of the training environments is due to their more challenging configurations. In both of these configurations the slot to the target wall is oriented in a different direction, so a robust policy needs to reason about this change. We observe that our proposed Expanded-MDP methods are able to perform well for this configuration. However, the end-effector action space shows a large variance *i.e.* many of the seeds fail to learn a robust policy to solve these configurations. Additionally, we also observe that 1-Ctrlr is unable to solve this task robustly. This shows the advantage of using multiple-controllers in parallel.

Figure 9.6a shows the different test configurations we evaluated. Each of the test configurations is a delta change in the wall lengths or angles from the train configurations. Table 9.9 shows the results on each of these test configurations. As seen above, our proposed Expanded-MDP formulations are able to outperform all other methods for all the configurations. 3-Priority performs well on most test configurations besides the slightly harder ones (E, G). This indicates that Expanded-MDP methods are able to learn more robust policies as compared to using a continuous priority score. Additionally, EE-Space perform poorly, especially for more different test configurations (E, F, G, H). Qualitatively, we observe that EE-Space policies often completely fail to generalize to the test configurations. 1-Ctrlr performs poorly on the D, E, and G, H configurations. For the initial two configurations, we observe that the learned policy can often get stuck around wall corners, which prevents it from completing the episode within the given time. Alternately, for the latter two environments, the learned policies across all seeds perform quite poorly, so they are not able to generalize to either of the test configurations.

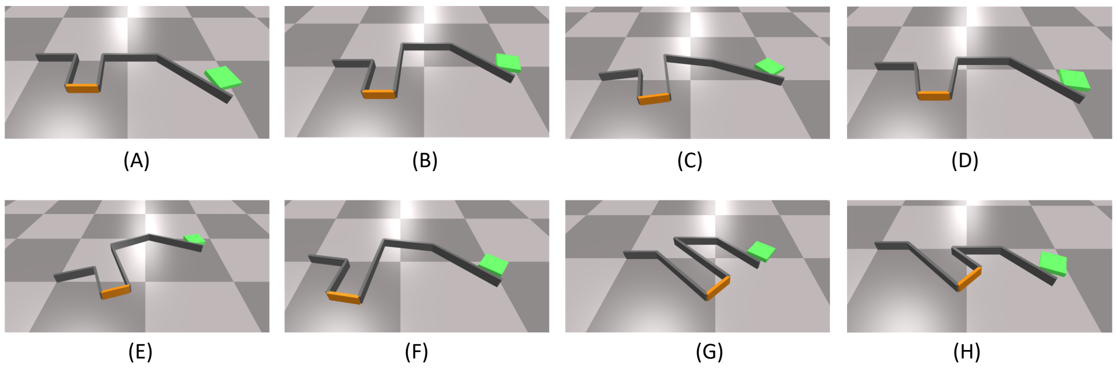


(a)



(b)

Figure 9.5: Different environment configurations used to train the Block Fit task. The plot below each environment configuration shows how the trained policy performed on each particular configuration.



(a)

Figure 9.6: Test configurations for the Block Fit task. Table 9.9 shows results on each environment configuration.

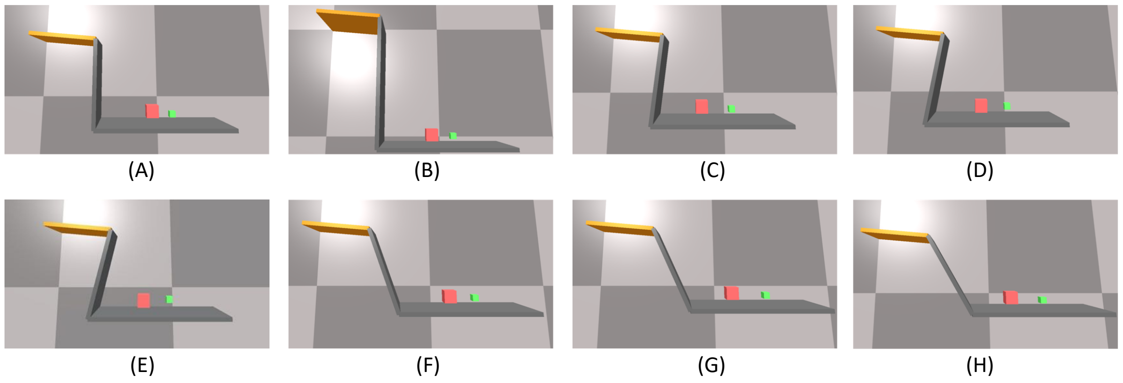
Test-Cfg	EE-Space	1-Ctrl	3-Pri.	3-Combo	3-Exp-Feat.	3-Exp-Single	3-Exp-Multi
A	0.86 (0.07)	0.98 (0.01)	**1.0 (0.0)**	0.19 (0.12)	**1.0 (0.0)**	**1.0 (0.0)**	**1.0 (0.0)**
B	1.0 (0.0)	1.0 (0.0)	**1.0 (0.0)**	0.20 (0.14)	**1.0 (0.0)**	**1.0 (0.0)**	**1.0 (0.0)**
C	0.85 (0.27)	0.96 (0.03)	0.97 (0.03)	0.18 (0.08)	0.98 (0.01)	**0.99 (0.01)**	0.96 (0.01)
D	0.89 (0.14)	0.71 (0.31)	**1.0 (0.03)**	0.165 (0.06)	**1.0 (0.0)**	**1.0 (0.0)**	**1.0 (0.01)**
E	0.16 (0.23)	0.64 (0.16)	0.76 (0.16)	0.0375 (0.05)	0.97 (0.02)	**0.99 (0.01)**	0.99 (0.02)
F	0.75 (0.39)	0.87 (0.16)	**1.0 (0.0)**	0.20 (0.14)	0.98 (0.01)	**1.0 (0.0)**	**1.0 (0.0)**
G	0.17 (0.30)	0.08 (0.23)	0.75 (0.45)	0.02 (0.03)	0.89 (0.13)	0.82 (0.23)	**0.90 (0.15)**
H	0.50 (0.53)	0.0 (0.0)	**1.0 (0.0)**	0.27 (0.18)	**1.0 (0.0)**	**1.0 (0.0)**	**1.0 (0.0)**

Table 9.9: Block Fit mean success on test environment configurations. Parentheses denote standard deviation across 8 seeds.

Block Push

Figure 9.7 shows the success rate for all the different environment configurations used in the Block Push task. As seen in the above figure, both 1-Ctrlr and 3-Priority show large variance in training performance. This is because both methods fail to learn the task for some seeds. All the Expanded-MDP methods are able to successfully complete the task without large variance. One reason for this is that a robust task strategy requires the use of multiple object-axis controllers to move the object along the vertical wall as well as to move it around the corner of the top wall. Although it is feasible to accomplish the task by quickly switching between controllers, it is hard to find a robust policy relying under such a switching mechanism, especially when controllers are being run for fixed number of steps. Additionally, while EE-Space also solves all the different environment configurations, its sample complexity is worse than the proposed Expanded-MDP methods.

Table 9.10 evaluates the learned policy on 8 different test configurations. Figure 9.8a plots each of these test configurations. These test configurations involve small perturbations in either the wall length or the wall angles (or both) from the train configurations. Specifically, we limit small perturbations to be a max change in vertical wall angle of $3 - 5°$ (A, C, D, F), while larger perturbations are sampled from $\sim\ 6 - 10°$ (B, E, G, H). We observe that EE-Space is usually robust to small perturbations of the environment, while slightly larger perturbations can significantly affect its performance. However, even with small perturbations our expand-MDP based methods are able to outperform the end-effector space. This shows the advantage of using a structured action space for learning, as Expanded-MDP methods perform well across both sets of configurations. Notably, the proposed approach only performs poorly on B and E configurations. For both configurations, as the policy pushes the red block up the middle wall, the agent block (green block) can sometimes end up under the red block, which leads to the green block falling, ending the episode. Additionally, both 1-Ctrlr and 3-Priority perform poorly on the test configurations. This is due to the poor performance of some of the learned policies (across a few seeds) on the train configurations. However, good train performance does not necessarily lead to good test performance. Specifically, 1-Ctrlr can move the green block upwards (by using the position controller for the top-wall), but it is often not able to robustly push it around the corner. This shows the advantage of being able to choose multiple object-axis controllers at each step.



(a)

Figure 9.8: Test environment configurations for the Block Push task. Table 9.10 shows results on each environment config.

Config	EE-Space	1-Ctrl	3-Priority	3-Combo	3-Exp-Feat.	3-Exp-Single	3-Exp-Multi
A	0.94 (0.06)	0.62(0.47)	0.43 (0.47)	0.0 (0.0)	0.97 (0.02)	0.98 (0.01)	**0.99 (0.00)**
B	0.27 (0.28)	0.27(0.30)	0.38 (0.43)	0.0 (0.0)	0.50 (0.27)	**0.72 (0.18)**	0.65 (0.30)
C	0.86 (0.23)	0.30 (0.40)	0.43(0.37)	0.0 (0.0)	0.91 (0.10)	**0.97 (0.01)**	**0.97 (0.04)**
D	0.70 (0.28)	0.07 (0.13)	0.42 (0.46)	0.0 (0.0)	0.89 (0.06)	**0.93 (0.07)**	0.88 (0.12)
E	0.48 (0.31)	0.01 (0.01)	0.36 (0.39)	0.0 (0.0)	**0.79 (0.17)**	0.69 (0.23)	0.67 (0.17)
F	**0.96 (0.03)**	0.73 (0.41)	0.38 (0.42)	0.0 (0.0)	0.88 (0.11)	0.95 (0.06)	**0.96 (0.03)**
G	0.89 (0.10)	0.67 (0.49)	0.35 (0.39)	0.0 (0.0)	**0.97 (0.03)**	0.92 (0.06)	0.89 (0.07)
H	0.61 (0.26)	0.27 (0.41)	0.34 (0.38)	0.0 (0.0)	0.79 (0.11)	0.78 (0.10)	**0.88** (0.07)

Table 9.10: Block Push mean success on test environment configurations. Parentheses denote standard deviation across 8 seeds.

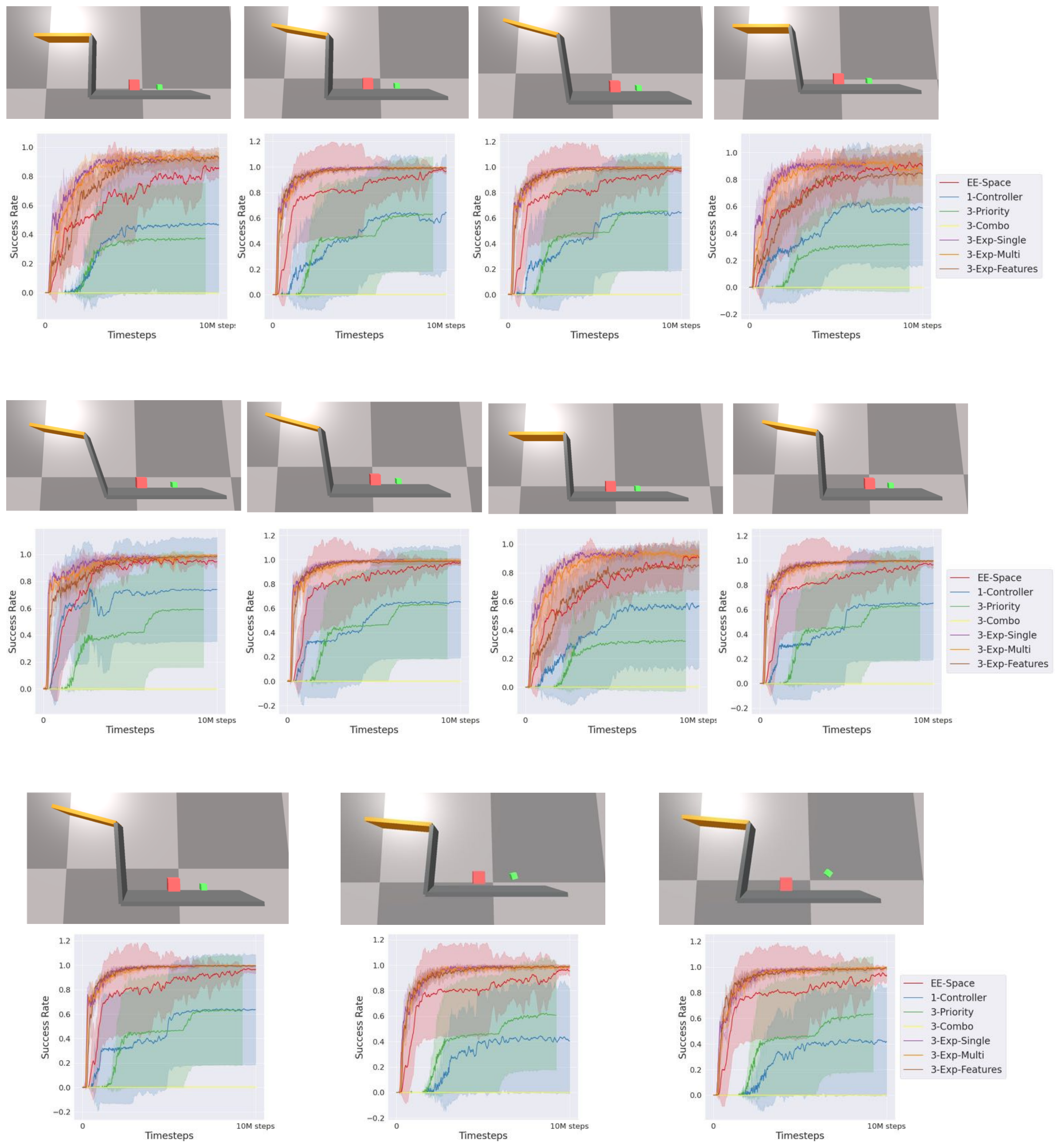
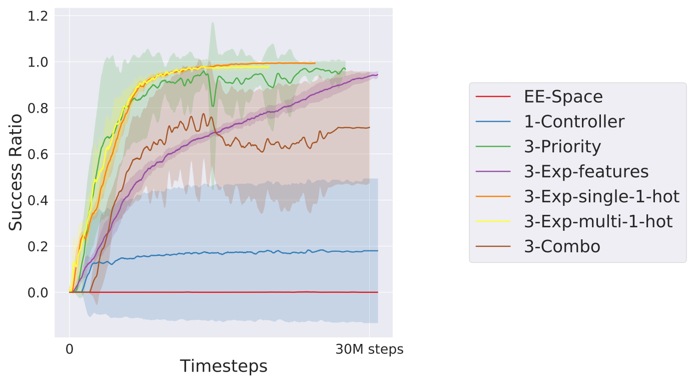
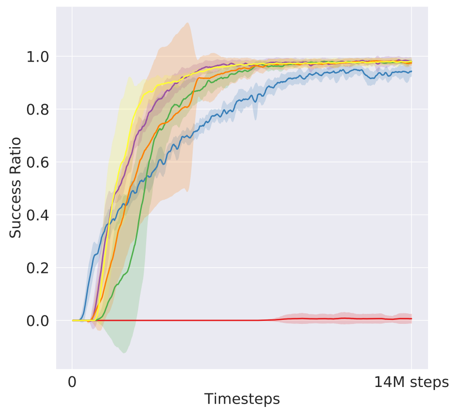


Figure 9.7: Different environment configurations used to train the Block Push task. The plot below each environment configuration shows how the trained policy performed on each particular configuration.

Franka Hex-Screw



(a) Franka Hex-Screw Task



(b) Franka Door-Opening Task

Figure 9.9: Franka tasks success ratios on training environment configurations during training.

Config	EE-Space	1-Ctrl	3-Priority	3-Combo	3-Exp-Feat	3-Exp-Single	3-Exp-Multi
0.7	0.0 (0.00)	0.05(0.14)	0.69 (0.44)	0.45 (0.43)	0.95 (0.04)	**0.97 (0.02)**	0.96 (0.035)
0.8	0.0 (0.00)	0.11(0.17)	0.66 (0.49)	0.43 (0.43)	0.97 (0.05)	0.96 (0.03)	**0.98 (0.01)**
1.1	0.0 (0.00)	0.21(0.37)	0.63 (0.49)	0.43 (0.36)	0.96 (0.03)	**0.98 (0.03)**	0.97 (0.03)
1.2	0.0 (0.00)	0.07 (0.15)	0.57 (0.42)	0.44 (0.42)	**0.97 (0.04)**	0.95(0.03)	0.95 (0.04)
1.4	0.0 (0.00)	0.0 (0.00)	0.67 (0.48)	0.34 (0.36)	0.92 (0.03)	0.94 (0.04)	**0.95 (0.03)**
1.5	0.0 (0.00)	0.03 (0.14)	0.54 (0.46)	0.24 (0.36)	**0.92 (0.04)**	0.90 (0.05)	**0.92 (0.03)**

Table 9.11: Franka Hex-Screw mean success across all test environment configurations. Parentheses denote standard deviation across 8 seeds.

Figure 9.9a plots the mean success rates for all the different approaches (including using controller features) during training. Since performance on all three train configurations (wrench and screw sizes) is very similar, we report one plot which averages the result for all the configurations. As seen in Figure 9.9a, EE-Space is not able to learn the task. While EE-Space policies can bring the wrench close to the screw, it does not achieve proper alignment and insertion, nor does it apply sufficient downard force, all of which are necessary to accomplish the task. Similarly, 1-Ctrlr also performs poorly. This is expected, since the task requires the use of multiple controllers *i.e.* force or position controller into the screw object while also rotating the wrench simultaneously. For approaches that use multiple object-axis controllers together, we find that the expand-MDP approaches perform the best, robustly learning the task each time. All the other approaches suffer from large variance in task performance.

Table 9.11 visualizes the result for each of the 6 different test configurations. Each test configuration uses a different wrench and screw scale. Our proposed approach is able to generalize to the different test configurations, achieving $\geq$ 0.9 success rate for all configurations. Although 3-Priority performs well in training, its test performance is slightly poorer. This is because some of the learned policies (seeds) fail to generalize well to any of the test configurations, while the remaining seeds perform as well as our

Expanded-MDP approaches. This variance in performance of the learned policies leads to lower mean success rate for 3-Priority. 1-Ctrlr fails to work well on any of the test configurations, which is expected given its poor training performance.

Franka Door-Opening

Figure 9.9b shows the average success rate in all train environments for the Door-Open. All methods except EE-Space are able to learn this task. One reason for this is that object-centric controllers make exploration in this task much more efficient than directly using the end-effector space. Although the EE-Space policy is able to grasp the handle, it fails to turn and pull. Table 9.12 shows quantitative results on test environments. Methods that use 3 controllers have very similar performance and perform better than 1-Ctrlr. Using multiple controllers is beneficial for this task. When turning the handle, the robot needs to learn to rotate the gripper and press down at the same time; when opening the door, the robot needs to simultaneously press the handle and pull it open. Since the reward function contains separate rewards for approaching the handle, turning the handle, and opening the door, the performance differences are due to the complexity of the task and not a lack of informative reward signals.

Config	EE-Space	1-Ctrl	3-Priority	3-Combo	3-Exp-Feat	3-Exp-Single	3-Exp-Multi
A	0.13 (0.13)	0.87(0.06)	0.93 (0.08)	0.97 (0.01)	**0.99 (0.01)**	**0.99 (0.01)**	**0.99 (0.01)**
B	0.00 (0.00)	0.96 (0.02)	**0.99 (0.01)**	**0.99 (0.01)**	**0.99 (0.01)**	**0.99(0.01)**	**0.99 (0.02)**
C	0.00 (0.00)	0.93 (0.03)	0.99 (0.01)	0.99 (0.01)	**1.00 (0.00)**	0.99 (0.01)	0.99 (0.01)

Table 9.12: Franka Door-Opening mean success on test environment configurations. Parentheses denote standard deviation across 8 seeds.

9.2.6 Controller Selection Analyses

We perform an ablation study to better understand the effects of algorithmic choices in our proposed approach. First, we analyze the effects of controller selection frequency, *i.e.*, we analyze the effect of T, where T is the number of steps for which object-axes controllers are run before the RL policy is queried again. Second, we qualitatively evaluate the learned controller selection policy by visualizing the learned policies. For both of these settings we use the Block Fit task.

Controller Selection Frequency. We evaluate how the controller selection frequency T affects the learning performance. For all previous experiments we use $T = 10$, *i.e.*, the object-axes controllers are run only for a few (10) steps. Although switching controllers frequently allows the RL policy to be more expressive, this comes at the associated cost of higher sample complexity. In this experiment we evaluate learning performance when controllers are allowed to run for much larger steps *i.e.* $T = 80$. To keep the overall simulation time fixed, we simultaneously reduce the maximum number of steps the RL policy is run, *i.e.* we reduce the episode length of the MDP T_{MDP}. This is important since running both the controllers and the RL policy for large number of steps is computationally prohibitive, since the total number of steps taken in the simulator is $T \times T_{\text{MDP}}$. We set $T_{\text{MDP}} = 15$ for this experiment.

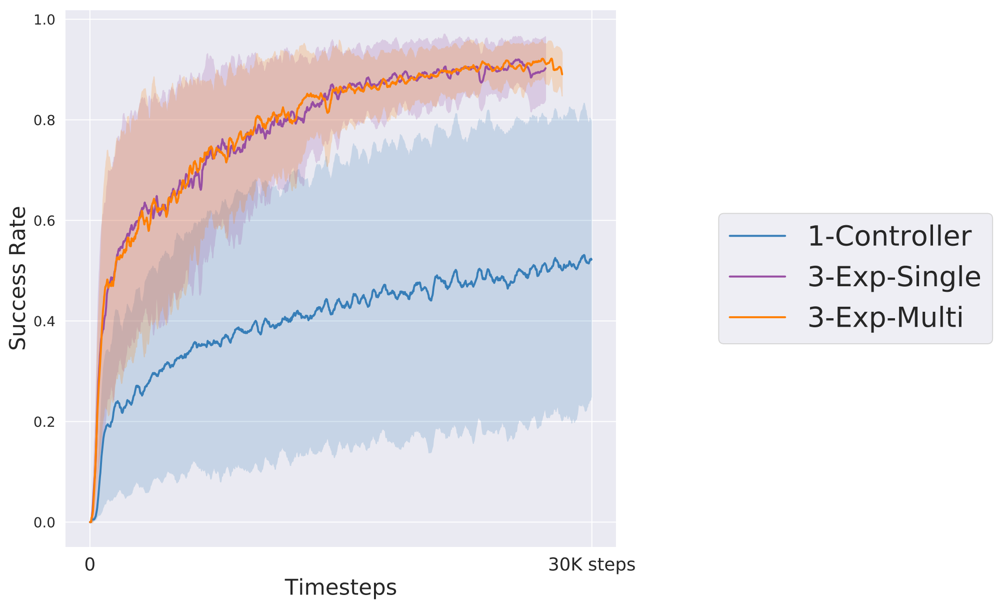


Figure 9.10: **Controller Selection Frequency:** Success rate for Block Fit task when object axes-controllers are run for $T = 80$ steps. Results averaged over 4 seeds (instead of usual 8).

Figure 9.10 plots the average success rate for all train configurations on the Block Fit task with $T = 80$. As seen above, our proposed expand-MDP based approaches are able to perform quite well. Alternately, selecting only one-controller (1-Controller) at each time step performs poorly as compared to $T = 10$ (Figure 9.5). This shows the advantage of being able to use multiple object-axes controllers in parallel. With small T the 1-Controller policy is able to complete tasks by quickly switching between different controllers. Since this is not possible with a larger T value, its performance decreases. This emphasizes the importance of using multiple-controllers in parallel. Additionally, Figure 9.10 shows that the Expanded-MDP based approaches are able to learn to perform the task in $30K$ steps only. This is significantly better than the $\sim 10M$ steps required for $T = 10$ (Figure 9.5).

Controller Selection Visualization

Figure 9.11 plots the controllers the policy selects along the block trajectory for two different train configuration of Block Fit. For the highest priority controller, the policy tends to select the one that attracts the block toward the target wall. Interestingly, the second priority controllers are associated with a different wall, i.e the left most wall. This shows that the RL policy learns to combine controllers across different objects (walls). For the initial part of the trajectory, the RL policy learns to rotate (priority 0) and move (priority 1) the block simultaneously. This composition of different behaviors is important for the policy to accomplish the task as fast as possible. In addition, the policy chooses from a few set of controllers for both priority 0 and priority 1, while it chooses from a large set of controllers for priority 2. This is because many different choices for the priority 2 controller would often have little to no effect, *e.g.* if both priority 0 and 1 controllers are position or force controllers, then choosing an additional position or force controller for priority 2 will likely have no effect. Thus, it is hard for the policy to learn the appropriate effect for lower priority controllers.

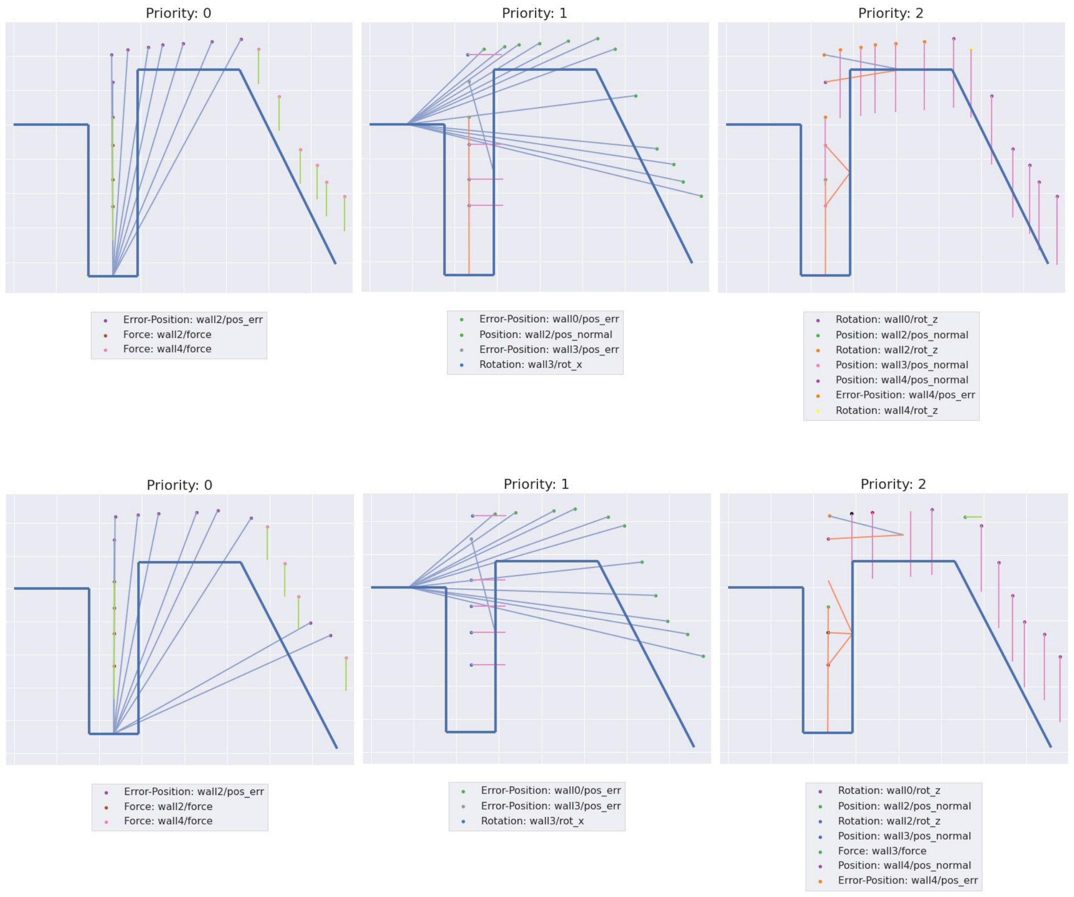


Figure 9.11: **Controller Selection Visualization** for Block Fit during Task Execution. The thick blue lines show the different walls in the environment. The dots represent the block position at each step. While the arrows represent the wall object used by the selected controller. The left most plot shows the top priority (priority: 0) controller being selected, while the right most plot shows the controllers with lowest priority (priority: 2). Top and bottom rows are two different train configurations (A and B from Figure 9.6a).

9.3 Appendix - Planning with Learned Skill Effect Models for Lifelong Robotic Manipulation

9.3.1 Related Works

	Low-level Skill					High-level Planner		
Paper	Policy	PC	Effects	Parameters	PS	Type	Init Plan	Heuristics
[98]	MP	n/a	Sim	n/a	S	MHA*	n/a	H
[230]	MP	n/a	Sim	n/a	LE	MCTS	L	H
[168]	MB	n/a	Sim	H, N-SG	S	Greedy	n/a	H
[134]	MF	n/a	L	n/a	S	MPPI	n/a	H
[161]	MF	L	SG	L, SG	LE	CEM	H	H
[252]	MF	n/a	L	L, N-SG	LE	CEM	H	L
[10]	MF	L	SG	n/a	S	RRT	n/a	n/a
[137]	IL	n/a	SG	L, SG	S	RL	n/a	L
[70]	IL	n/a	SG	L, SG	S	Sampling	L	n/a
[125]	IL	n/a	L	L, N-SG	LE	MPC	n/a	H
[22]	HC	H	H	n/a	S	WA*	n/a	H
[222]	HC	H	SG	L, SG	V	Sampling	H	n/a
[254]	HC	L	L	H, N-SG	LE	MPC	L	H
[89]	HC	H	H	H, N-SG	S, Sym	HPN	n/a	H
[104]	HC	L	L	n/a	Sym	mGPT	n/a	n/a
[247]	HC	L	SG	H, N-SG	S, Sym	PDDLStream	n/a	n/a
Ours	HC	H	L	H, SG, N-SG	S	WA*	n/a	H

Table 9.13: Related works on task planning with skills.

Table 9.13 compares the discussed related works and our proposed approach. We include this table to illustrate the subtle but important similarities and differences among prior approaches, which may be overlooked when they are compared at a more general level. This is not an exhaustive list of all works in relevant areas. See below for the list of acronyms of each column. While we list the skill policy and preconditions as hardcoded in our work, this is not a requirement, as our task planner does not restrict the specific implementations of skills.

Policy: **MB** - Model-Based Optimization, **MF** - Model-Free RL, **MP** - Motion Primitives, **HC** - Hardcoded Controllers, **IL** - Learned via Imitation Learning

Preconditions (PC): **H** - Hardcoded, **L** - Learned

Effects: **H** - Hardcoded, **Sim** - Simulator, **L** - Learned (usually from simulator data), **SG** - Assumes skill reaches subgoal, **FC** - Fixed by algorithmic skill construction

Parameters: **H** - Hardcoded , **L** - Learned (e.g. training a CVAE parameter sampler), **SG** - Subgoal, **N-SG** - Not Subgoals

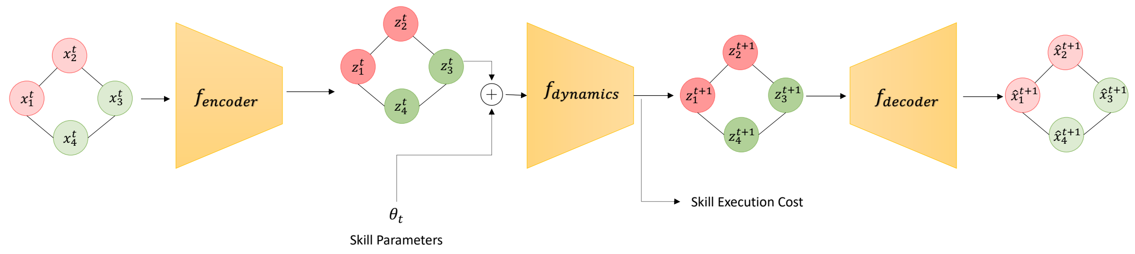


Figure 9.12: Latent Space Model Architecture. The encoder model is used to encode states into latent states while the decoder model converts latent states into original states. While the skill dynamics model acts on the latent state and given skill parameters outputs the final latent state and a skill execution cost. We use a shared encoder-decoder model for all skills while each skill has a separate dynamics model.

Planning State (PS): **S** - State (e.g. low-dimensional object features), **V** - Visual (e.g. images, point clouds), **LE** - Latent Embeddings (usually learned to encode high-dimensional visual observations), **Sym** - Symbols

Init Plan (if the planner uses an initial plan, where does it come from?): **L** - Learned (e.g. from experience or predefined plan skeleton), **H** - Hardcoded (e.g. a plan skeleton of a skill sequence, linear interpolation of subgoals, or randomly sampled subgoals)

Heuristics (e.g. task value functions or shaped rewards/costs that guide planning towards a goal): **H** - Hardcoded, **L** - Learned (could be from experience, analytical models, or demonstrations)

9.3.2 SEM Implementation

SEM Implementation: The GNN model contains four learnable multilayer perceptron (MLP) modules. The first is node feature embedding module, with two layers of sizes $[32, 32]$. The second is a message embedding module that pass messages among nodes. It has three layers of sizes $[128, 128, 128]$. The third is a node-level prediction module, with two layers of sizes $[64, S + 32]$, where S is the dimension of per-object features, and the extra 32 is used to produce graph-level predictions. The fourth is a graph-level prediction module, which takes as input the sum of the last 32 dimensions of node-level predictions and passes it through an MLP with sizes $[32, 1]$ to predict skill execution costs.

The loss weight for terminal state prediction is $\lambda_s = 100$, and the weight for cost prediction is $\lambda_c = 1$. All non-linearities are ReLU. The network is trained with the Adam optimizer with a batchsize of 128, initial learning rate of 0.01, and for 300 epochs on every training run.

Each input object feature contains the position of the block, its color, and its index. Index is needed for SEMs to identify which block is being grasped for *Pick-Place*. Color and indices are encoded with an offset positional encoding. For example, for the ith color or index, its encoding is a two-dimensional feature vector $[i, i + 1]$. We found in practice that this simple approach allowed our network to capture indices well and also allowed scaling to different number of objects and colors, which cannot be done with one-hot encoding.

Iterative Training: Instead of collecting all the data from the planner for training the SEM models, we use heuristics to sample the more relevant data. First, we bias data collection towards longer paths in the planning tree since these paths are more likely to be closer to the relevant task. Second, we also bias data collection towards newly added skills. Since in the beginning we would not have sufficient data for the newly added skill, this ensures that we get sufficient data for such new new skills. Algorithm 1 lists the pseudo-code for the data-collection procedure including the heuristics used to sample paths from the planner.

Algorithm 1 SEM iterative training pseudocode.

```
Input: Set of skills $o$s, set of training tasks $\tau$s.
  for all new skills $o$ do
    Sample $M_0$ initial states from $x_0 \sim p(x_0)$.
    Sample $P_0$ parameters $\theta$ for each initial state that satisfies $\beta_o(x_0, \theta)$.
    Simulate $\pi_o$ with all $M_0 \times P_0$ (state, parameter) tuples.
    Add to dataset: $\mathcal{D}_o \leftarrow \{(x_0, \theta, x_T, c_o)\}$.
  end for
  for all skills $o$ do
    Train on $D_o$ for $E$ epochs the skill's SEM from scratch or fine-tune previous model if it exists.
  end for
  for all training tasks $\tau$ do
    Sample $M_p$ initial states from $x_0 \sim p(x_0)$.
    for all sampled initial state $x_0$ do
      $G \leftarrow$ get planning graph by running WA* on $\tau$ with max search depth $N_d$, max node expansions
      $N_e$, and timeout $T_p$.
      Sample $N_l$ nodes in $G$'s open list.
      For each of the $N_l$ nodes, trace their optimal path found so far $P$ from $x_0$.
      Give each of the $N_l$ paths a weight $w = n_o + 10n_s$, where $n_o$ is the number of old skills in the
      path, $n_s$ the number of new skills.
      Sample without replacement $N_s$ paths from all $N_l$ paths with their normalized weights as likeli-
      hoods.
      Simulate each $N_s$ path.
      Add skill transitions to the corresponding dataset $D_o$ while ignoring duplicates (e.g. some paths
      may share identical initial segments).
    end for
  end for
```

Weighted A* Planner: Weight A* (WA*) is a best-first graph search algorithm that expands nodes in the order of lowest $g(x) + \epsilon h(x)$, where $g(x)$ is the total cost of the current optimal path from the initial node to x, and $h(x)$ is a heuristic function. The hyperparameter ϵ is called the inflation factor and is usually greater than or equal to 1. If it is set to 1, then the search is no different from A*. The higher the ϵ above 1, the greedier the search becomes at following the heuristic.

Hyperparameters: For all blocks in bin tasks (tasks *A* and *C*) with just the *Pick-Place* skill we use a high value of $\epsilon = 20$ since there are not many ways to achieve the task. With the addition of more skills i.e, *Tray-Slide*, *Tray-Sweep* and *Bin-Tilt* we set $\epsilon = 2$. We found that this value to be sufficient to choose optimal plans. Additionally we also set the maximum search depth for both tasks A and C to be 8, since the longest plan for these tasks should be of length 8. For the colored blocks in bin tasks (tasks B and D), we used $\epsilon = 2$ and a max search depth of 5. Additionally, we sample a larger number of parameters for *Pick-Place* skill as compared to the other skills. This is because the *Pick-Place* skill affects each block separately and thus to find optimal plans we need to sample sufficient parameters for each block. This necessitates a larger number of parameters for this skill. For *Pick-Place* we sample 24 parameters while for all other skills we sample $[4, 6]$ parameters.

9.3.3 Guarantees on the Constructed Graph

WA* guarantees completeness and bounded suboptimality on a given graph. Here we show that under smoothness assumptions, the graph constructed with our parameter sampling approach is suitable for search-based planning. Theorem 1 is about completeness — the distance between the reached goal state by any solution path and the closest last state of a path on the graph is bounded. Theorem 2 is about solution quality — for any solution path with a certain cost, the graph will contain a solution path with a bounded cost difference.

Definition 1. *The dispersion [118] of a finite set A of samples in a metric space (X, ρ) is defined as*

$$\delta(A) = \sup_{x \in X} \min_{p \in A} \rho(x, p)$$

Intuitively, it is the radius of the largest empty ball that can be drawn around any point in X without intersecting any point in A.

In the following discussion, we assume all preconditions and the goal set are open sets.

Theorem 1. *Let the skill transition function $f_o : \mathcal{X} \times \Theta \to \mathcal{X}$ be Lipschitz continuous with a maximum Lipschitz constant K and $\eta = (x_0, o_0, \theta_0, x_1, \cdots, x_N)$ be a solution to the planning problem. Then, $\exists$ a path $\eta' = (x_0, o_0, \theta'_0, x'_1 \cdots, x'_N)$ on the constructed search graph such that $||x_N - x'_N|| \leq 2\delta\kappa_N = 2\delta\frac{K(K^N-1)}{K-1}$ if the dispersion δ of parameter samples is small enough.*

Proof Consider an instance of the randomly generated search tree $\mathcal{T}$ of depth more than N. We pick a small enough δ such that $\mathcal{T}$ contains at least one path η' that has the same sequence of skills as η. However, due to random sampling, the sampled parameters may not be the same. For every θ_i at a state, $\exists$ a parameter sample θ'_i such that

$$||\theta_i - \theta'_i|| \leq 2\delta \tag{9.11}$$

Using the definition of Lipschitz continuity and the triangle inequality, we have

$$\begin{aligned} ||f_{o_i}(x_i, \theta_i) - f_{o_i}(x'_i, \theta'_i)|| &\leq K||(x_i, \theta_i) - (x'_i, \theta'_i)|| && (9.12) \\ &\leq K||x_i - x'_i|| + K||\theta_i - \theta'_i|| && (9.13) \end{aligned}$$

In particular,

$$||x_1 - x'_1|| = ||f_{o_0}(x_0, \theta_0) - f_{o_0}(x_0, \theta'_0)|| \leq K||\theta_0 - \theta'_0||$$

and

$$\begin{aligned} ||x_N - x'_N|| &= ||f_{o_{N-1}}(x_{N-1}, \theta_{N-1}) - f_{o_{N-1}}(x'_{N-1}, \theta'_{N-1}|| \\ &\leq K||\theta_{N-1} - \theta'_{N-1}|| + K||x_{N-1} - x'_{N-1}|| \\ &\leq \sum_{i=0}^{N-1} K^{N-i}||\theta_i - \theta'_i|| \leq \Delta\theta \sum_{i=0}^{N-1} K^{N-i} \\ &\leq 2\kappa_N\delta \text{ (Using inequality 9.11)} \end{aligned} \tag{9.14}$$

where $\kappa_N = \frac{K(K^N - 1)}{K-1}$.

To ensure that we have a path η' on the graph that terminates ϵ-close to the terminal state of η, we require that $||x_N - x'_N|| \leq 2\kappa_N\delta \leq \epsilon$.

To guarantee that η' will also be a solution, we additionally need to ensure that $\epsilon < r$, where r is the radius of the largest ball we can draw around x_N in G (goal set). This is always possible if G is an open set.

Theorem 2. *Let the cost function be Lipschitz continuous with a maximum Lipschitz constant of L. Let η be a solution path of cost $c(\eta)$ and $r > 0$, then for a sufficiently small dispersion δ, $\exists$ a solution path η' on the constructed search graph with cost $c(\eta') \leq c(\eta) + \delta NL[1 + \sum_{i=1}^{N} \kappa_i]$.*

Proof Choose δ such that $2\kappa_N\delta < r$ and $\exists$ a path η' on the graph with the same sequence of skills as η. Then, from the previous theorem, $\exists$ a path η' on the graph such that $||x_N - x'_N|| \leq 2\kappa_N\delta < r$. By the definition of r, $x'_N \in G$ and hence η' is a solution path. Next, we bound the cost of this path.

The cost of a path $c(\eta) = \sum_{i=0}^{N-1} c_{o_i}(x_i, \theta_i)$. For convenience, here we write the cost function as skill-specific cost functions that vary w.r.t the initial state and skill parameters, instead of step-wise costs on state and low-level controls. Let $c_i = c_{o_i}(x_i, \theta_i)$ and $c'_i = c_{o_i}(x'_i, \theta'_i)$. Then, using the definition of Lipschitz continuity, we have

$$\begin{aligned} ||c_i - c'_i|| &\leq L||(x_i, \theta_i) - (x'_i, \theta')|| \\ &\leq L||x_i - x'_i|| + L||\theta_i - \theta'_i|| \\ &\leq L[2\kappa_i\delta + \delta] \end{aligned} \tag{9.15}$$

Hence,

$$\begin{aligned} ||c(\eta) - c(\eta')|| &= ||\sum_{i=0}^{N-1} c_i - c'_i|| \leq \sum_{i=0}^{N-1} ||c_i - c'_i|| \\ &\leq \delta NL[1 + 2\sum_{i=1}^{N} \kappa_i] \end{aligned} \tag{9.16}$$

The dispersion of a set of sampled parameters depends on the sampling strategy used. For uniformly random sampling, we can only estimate it probabilistically. [66] prove bounds on the expected dispersion of a set of i.i.d points which tightens with more points, i.e., we are more confident of a lower expected dispersion with a larger number (B_o) of parameter samples.

Theorem 2 provides two additional observations. First, skills with dynamics that don't change fast, i.e., their Lipschitz constant are small, can be approximated with a sparse graph (less parameter sampling). Second, longer horizon tasks require more parameter sampling to guarantee good quality plans.

9.3.4 Task Domain Skill Details

All skill policies are implemented by end-effector waypoint following, where trajectories between waypoints are computed via min-jerk interpolation, and low-level control is achieved by end-effector impedance control. A subset of waypoints for each skill is indirectly determined by the skill parameters. For example, for *Pick-Place*, the object index parameter determines which object to pick. Together with the current state, they are used to compute the waypoints associated with the picking motion.

Pick-Place: This skill picks up a chosen block and places it at a target location.

Parameter. A 4-vector consisting of an index corresponding to which block to pick and a 3D position for placement

Parameter Sampling. For which block to pick, we sample the index of a block on the table with collision-free grasps. For the placement position, we first sample which general placement region to use, table, bin, or tray, with probabilities $[0.2, 0.5, 0.4]$. Then, we randomly sample a position on the surface of reach placement region. The bin placement region is the half of the bin that is closer to the robot.

Precondition. The precondition function check returns satisfied if the placement position is collision free. It assumes the pick grasp is collision free and the placement location is reachable.

Tray Slide

This skill picks up the tray, brings it over the bin, rotates the tray to let blocks fall to the bin, then brings the tray back to its original pose.

Parameter. One value that determines where along the bin does the slide rotation motion. This allows dropping blocks over the close or far side of the bin.

Parameter Sampling. This is done via uniform sampling over a range of the length of the bin.

Precondition. The precondition is satisfied when there is at least one block on the tray.

Tray Sweep: This skill picks up the tray, rotates it $90°$, brings it down toward the table, sweeps downstream objects into the bin, and returns the tray to its original pose.

Parameter. One value that determines where along the table the sweep motion begins.

Parameter Sampling. This is done via uniform sampling over a range of the width of the table.

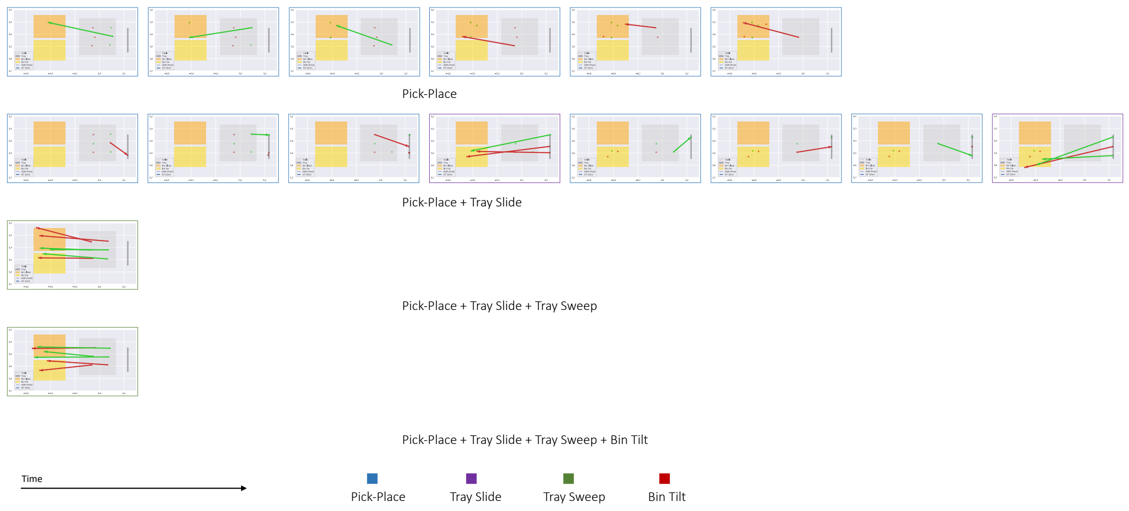


Figure 9.13: Qualitative results: Plans found for *Task-A* with increasing number of skills. Left most columns are the skills executed at t=0, with skills being executed as we move towards right. Each row also lists the different skills used to plan.

Precondition. The precondition is satisfied when there is at least one block that will be swept by the skill and the starting location for the tray is collision-free.

Bin Tilt: This skill lifts one side of the bin by a desired angle, allowing objects to slip down toward the far side of the bin, before return the bin to the original configuration.

Parameter. One value that determines the angle of the tilt. With a shallow angle it is possible for some blocks to remain in the near side of the bin, or not move at all due to friction.

Parameter Sampling. This is done via uniform sampling in the range of $[5°, 20°]$.

Precondition. The precondition is satisfied if there is at least one block in the bin.

9.3.5 Additional Results - Qualitative Planning Results

In this section we show qualitative results for our train and test tasks.

Figure 9.13 shows the qualitative results for the train task *A* with an increasing number of skills. In the above result we see that with only *Pick-Place* skill the planner finds a plan which needs to pick all blocks and place them in the bin. However, after adding the *Tray-Slide* skill (row 2) the planner finds a longer plan but with a lower cost 4.3. This lower cost is a result of being able to use the Tray-Slide skill to transport multiple blocks to the bin directly. Additionally, with the addition of *Tray-Sweep* skill we can perform the task with only 1 skill which significantly reduces the overall plan cost.

Similarly, Figure 9.14 and 9.15 show the qualitative results for both test tasks *B* and *D* respectively. Similar to before, with an increasing number of skills the overall plan and its associated cost changes significantly. For task *B* we observe that adding *Tray-Sweep* improves the performance only in some scenarios as shown

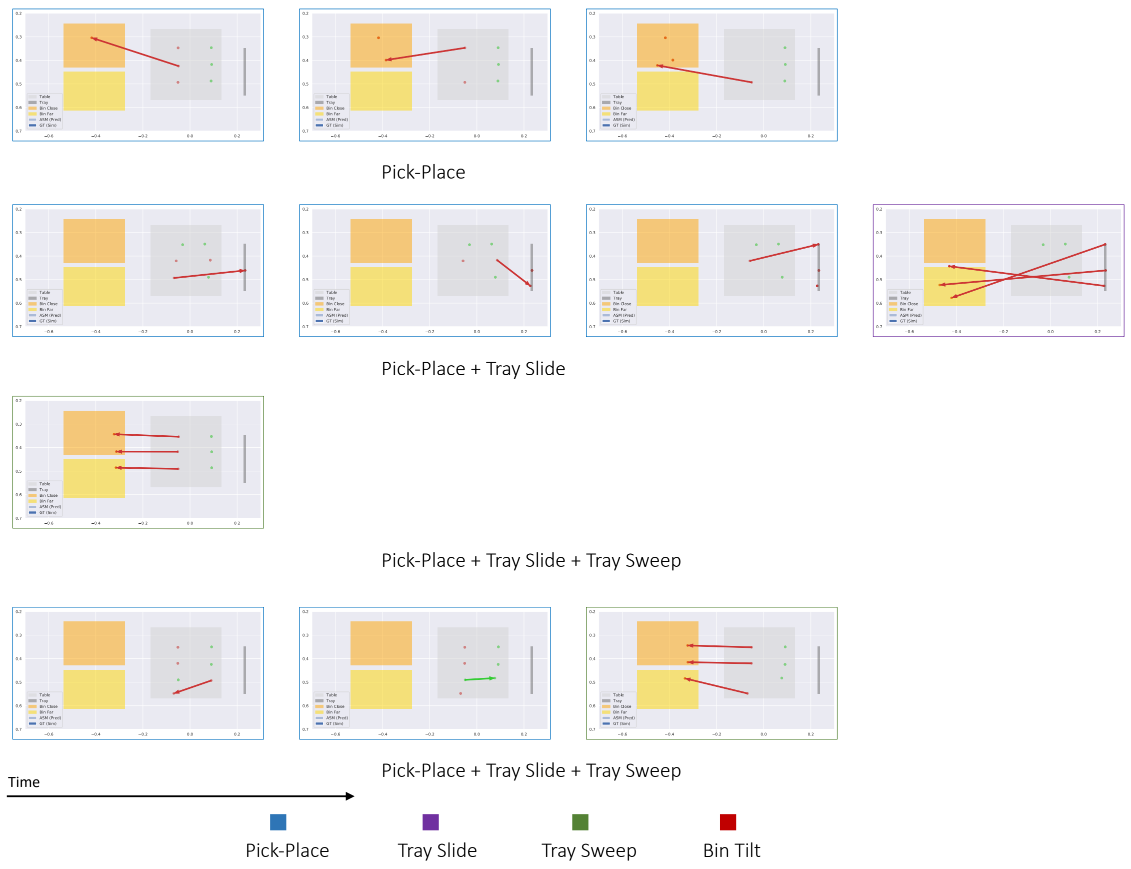


Figure 9.14: Qualitative results: Plans found for Task-B with increasing number of skills. Left most columns are the skills executed at t=0, with skills being executed as we move towards right. Each row also lists the different skills used to plan.

in Figure 9.14 (rows 3 and 4). In another instance the robot can also perform *Pick-Place* to align the red blocks to sweep them into the bin (row 4).

Similarly, for test-task *D* we observe that using *Pick-Place* alone is not sufficient (as observed in Figure 4.3). However, adding the *Tray-Slide* skill makes the task feasible (top row). However, in contrast to tasks *A* and *B*, adding *Tray-Sweep* does not affect task execution since the robot cannot move the blocks to the far bin with this skill. We observe this in Figure 9.15 where the top two rows use the same set of skills. However, adding the *Bin-Tilt* skill allows the planner to use the *Tray-Sweep* skill, as shown in Figure 9.15 (row 3, second row from bottom).

Video results for most tasks and settings can be found at the project page `https://sites.google.com/view/sem-for-lifelong-manipulation`.

9.3.6 Failure Modes

Here we discuss examples of planning and execution failures due to insufficiently trained SEMs.

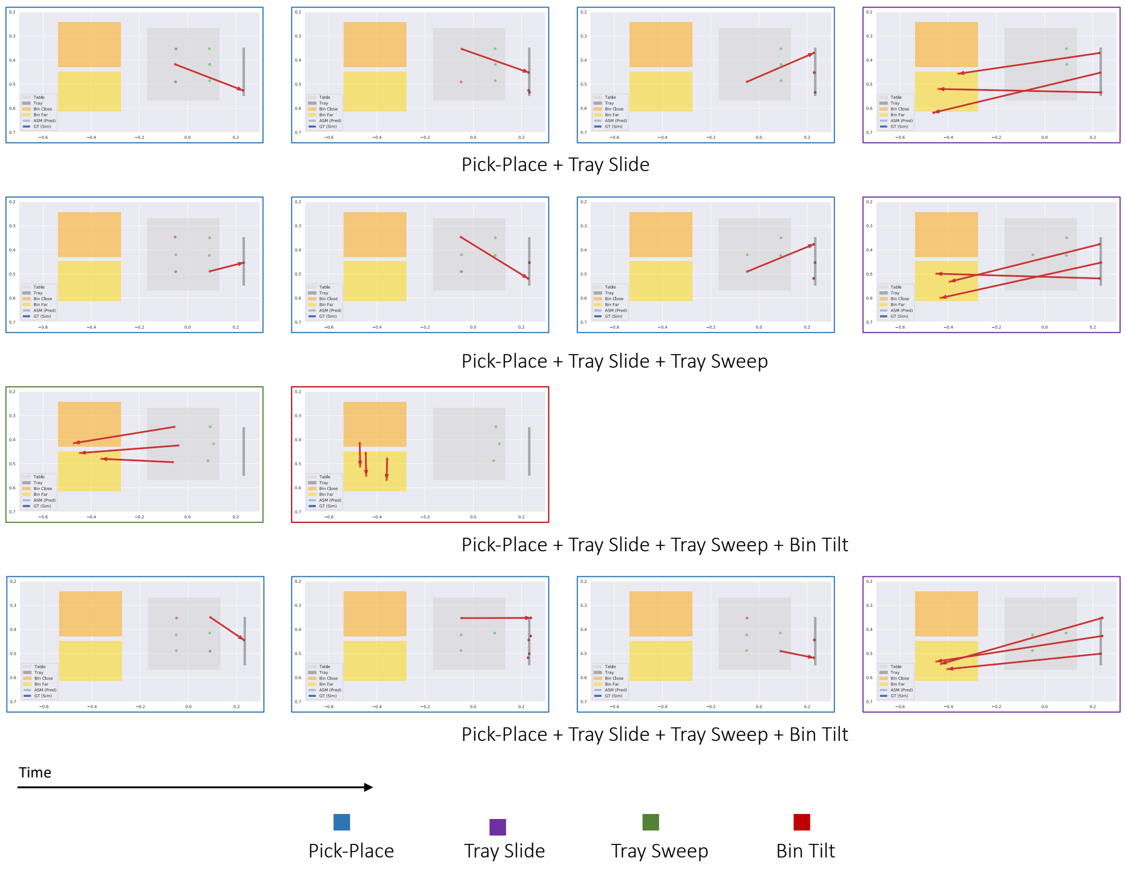


Figure 9.15: Qualitative results: Plans found for *Task-D* with increasing number of skills. Left most columns are the skills executed at t=0, with skills being executed as we move towards right. Each row also lists the different skills used to plan.

First, if SEM predictions are not sufficiently accurate, then the predicted states would do not satisfy the preconditions of necessary skills, and the planner will not find a plan. For instance, when *Tray-Sweep* is trained with less data, the learned SEM predictions can incorrectly predict that the blocks move to the edge of the table instead of being moved to the bin. Thus, the planner considers this to be a non-optimal action and instead resorts to using the remaining skills (*Pick-Place* and *Tray Slide*), which leads to a sub-optimal plan if one is found. Another scenario where incorrect SEM predictions lead to failure is when *Tray-Slide* is added. Given the previously used **Pick-Place** was mostly trained on moving blocks from the table to the bin (because of the train-task bias), the existing **Pick-Place** SEM may inaccurately predict that the skill is unable to move blocks to the tray. This results in the failure for *Tray-Slide* preconditions which finally results in either the planner timing out or finding sub-optimal plans.

Second, there are also scenarios where the planner is able to find a plan despite inaccurate SEM predictions, but the plan does not work in execution. One common instance is with *Pick-Place*, where early on with insufficient data the SEM often inaccurately predicts that the skill will move blocks to the bin, despite the target placement location being on the edge of the table. Given the incorrect prediction by the SEM, the planner mistakenly believes that the block has been placed in the bin, and it may find a plan but its execution does not lead to success.

9.4 Appendix - Adapting Pretrained Vision Models For Robotic Manipulation

9.4.1 Experimental Setup

We provide further training details for our approaches in the following sections. First we discuss the adapter parameterizations in more detail specifically detailing the adapter locations and their associated trainable parameters.

Adapter Parameterizations As noted previously in Sub-Section 6.3.2 we use *bottom*, *middle* and *top* adapters. We discuss the overall number of parameters for each of these adapter types. Table 6.4 provides an overall count for the number of adapter parameters.

Bottom: For NFNet and ResNet architectures we use the initial convolution, while for ViT we use the initial patch embedding as the only bottom set of layers. We add 1 adapter for this bottom layer. Since the image input only contains 3 channels and the bottom layer consists of only 16 and 256 channels for NFNet and ResNets respectively, bottom adapters require very few (< 1000) parameters. While for ViT since we have a large feature output, it results in a much larger number of parameters $\approx 0.7M$.

Middle: For middle adapters we use 4 and 6 adapters for the convnet and transformer based architectures respectively. For the convnet based architectures we add each adapter to the first convolution in each block group except the last group, where we add it at the end. While for the transformer based architectures we apply them at layers $\{0, 1, 5, 6, 10, 11\}$ which yields around $\sim 400K$ adapter parameters. While better choices and adapter placements may exist we use these uniformly across all tasks. Finally, the output of middle layer consists of a set of spatial features. For NFNet and ResNet these are output of layer 4 and final conv with a spatial size of 7×7 and a feature size of 3096 and 2048 respectively. For ViTs we get 196 patches each with 768 features.

Top: As noted previously in Sub-section 6.3.2, the high dimensional spatial features from the middle layer can be reduced either via mean pooling or by projecting them onto a lower dimensional space through a single layer. Since prior works use mean pooling we use it to compare our work with them. However, since manipulation tasks are spatial in nature we also investigate down-projecting the high dimensional spatial features into smaller dimensions and concatenating them. This formulation avoids any loss of spatial information. To achieve this for NFNet and ResNets we use 1 convolution layer with 1×1 kernel and 41 output channels. While for ViT we use a shared MLP that projects each patch embedding into a 20 dimensional feature. Finally, to further process these features we *optionally* add 2 MLPs each with 256 parameters, we refer to these as top layer adapters.

Policy Head: We universally use a linear policy head that converts the output of the top layer into the robot action to be executed.

9.4.2 Training Details

As noted in the main paper we use behavior cloning with mean squared loss as the optimization objective. We use a linear policy head to predict continuous actions. While specific action parameterizations, such as

Training Parameters	MetaWorld	Franka-Kitchen	RGB Stacking
Loss	MSE	MSE	MSE
Optimizer	Adam	Adam	Adam
Learning Rate	1e-4	1e-3	1e-4
Weight Decay	1e-6	1e-6	1e-6
Gradient Norm Clip	1.0	1.0	1.0
Training Steps	40K	40K	200K
Learning Rate Schedule	cosine	cosine	cosine
Learning Rate Schedule Warmup Steps	5K	5K	10K
Adapter Features Size	32	32	32

Table 9.14: Training Details for each of the three different task suites used in our work. For each task within the task suite we use the same set of hyperparameters.

the use of binary gripper can help increase performance, for a fair comparison of representations we avoid using any such techniques. Table 9.14 lists out the detailed hyperparameters used in our experiments. We uniformly use the same set of hyperparameters across most task settings except the learning rate, wherein we found using a slightly higher learning rate of $1e-3$ works better for Franka-Kitchen tasks.

Network Details: As discussed before our implementation uses three different network architectures – NFNets, ResNets and ViTs. Figure 6.3 presents the overall architecture. In settings where we use proprioceptive information, we use a single linear layer with 256 dimensions to map the low dimensional proprioceptive information to a higher dimensional space. This high dimensional proprioceptive information is then concatenated with the visual features before being forwarded to the 2 layer MLP (each with 256 units). Further, since we evaluate our approach across very different architectures we use the same policy form across all of them. Thus, we avoid using any normalization techniques such as BatchNorm or LayerNorm in our policy implementation.

9.4.3 Additional Results

We provide further results for the use of adapters in the three different environment suites considered in the main paper. We then discuss the ablation results on adapter locations for all suites and network architectures. For these results, in addition to average metrics across all enviroments, we also provide task specific metrics.

Adapter Results with Proprioceptive Information. In this section we present detailed task-specific success rate using our proposed adapters for each task in the three manipulation suites. For these results we use all top, bottom and middle adapters in our implementation. Further, in addition to visual features we also utilize proprioceptive information for these results. Table 9.15, 9.16 and Table 9.17 report task-specific results for MetaWorld, Franka-Kitchen and RGB Stacking suites using all three different architectures with imagenet pretrained weights. Comparing Table 9.15 with previous results in Table 6.2 we see that adding proprioceptive information results in $\approx 10\%$ increase in the average success rate. This increase holds consistently across all architectures. More interestingly we also find that for most tasks

	Assembly	Bin-Picking	Button Press	Drawer Open	Hammer	Average
NFNet	0.92	0.7	0.94	0.96	0.94	0.89
ResNet	0.90	0.66	0.96	0.94	0.92	0.88
ViT	0.92	0.8	0.91	0.98	0.92	0.91

Table 9.15: Task specific results for using bottom, middle and top adapters with proprioceptive information (proprio) for each task in MetaWorld.

	Knob1-On	LDoor-Open	Light-On	Micro-Open	SDoor-Open	Average
NFNet	0.46	0.44	0.72	0.32	0.94	0.58
ResNet	0.48	0.46	0.60	0.30	0.88	0.54
ViT	0.6	0.48	0.59	0.36	0.83	0.57

Table 9.16: Task specific results for using bottom, middle and top adapters with proprioceptive information (proprio) for each task in Franka-Kitchen suite.

(except Bin-Picking) the agent can reach greater than 90% performance, while for some tasks such as *Button-Press* and *Drawer-Open* it can even reach close to 100% performance.

Table 9.16 shows the results for each task in the Franka-Kitchen suite. Compared to previous results in Table 6.2 we see a much larger increase in the performance ($\approx$ 60% relative performance increase on average) of each architecture in the Franka-Kitchen suite. One reason for such a large increase is the very limited state space distribution for these tasks. Since all objects in the environment are fixed and only the initial robot configuration changes, it is much easier for the robot to memorize the proprioceptive information and map it to observed expert actions for improved task performance. Additionally, while both MetaWorld and Franka-Kitchen use 25 demonstrations, each demonstration in MetaWorld has 500 steps while in Franka-Kitchen each demonstration is only 50 steps. This results in $10\times$ difference in the amount of training data. However, since prior works use these settings for a fair comparison we follow similar evaluation protocols.

Effects of Adapter Locations. In this section we investigate the effect of inserting adapters in each of the different network layers as discussed in Subsection 6.3.2 and initially explored in Subsection 6.5.2. Due to space constraints in Subsection 6.5.2 we only provide results for the RGB Stacking task. In this subsection we show results across all manipulation suites and network architectures. For ease of comparison we also plot the RGB Stacking results from before.

Figure 9.16 shows results for inserting adapters in each network layer across all 3 task suites and architectures. As noted before, we split the results in each plot into two parts. 1) without using top layer adapters (i.e. directly using a linear policy head), and 2) using a top layer adapter (i.e. using 2 additional MLPs before the linear policy head). Moreover, in addition to the different adapter locations we also show results for fixed pretrained representations (Pretrain Feat.) and full fine-tuning (Full FT.) both with and without top adapters. As noted in the main paper, prior works always use such top adapters in their implementations.

	Triplet 1	Triplet 2	Triplet 3	Triplet 4	Triplet 5	Average
NFNet	0.40	0.18	0.11	0.67	0.90	0.45
ResNet	0.48	0.34	0.11	0.62	0.86	0.48
ViT	0.25	0.40	0.13	0.80	0.85	0.49

Table 9.17: Task specific results for using bottom, middle and top adapters for each task in RGB-Stacking suite.

Top Adapters: In our discussion in Subsection 6.5.2 we showed that for the RGB Stacking task top adapters are quite important to achieve close to optimal task performance. We note that the greyed out plots in Figure 9.16 indicate methods that do not use top adapters. Our results in Figure 9.16 show that this holds true for both MetaWorld and Franka-Kitchen suites as well. For both of these suites we find that using top adapters improves the downstream manipulation performance. However, as seen in the metaworld results (top row of Figure 9.16), full fine-tuning approaches (last bar in each plot) can reach good performance even without top adapters. However, this does not hold for the Franka-Kitchen tasks (middle row in Figure 9.16). We hypothesize this is because of the metaworld setup, wherein there is usually a single object centered on an otherwise empty table, which presents an easier visual setting and simply fine-tuning the high capacity pretrained visual model can extract the appropriate task representation. However, we do note that our use of adapters is able to closely match the full fine-tuning performance across all architectures.

Bottom Adapters: Similar to RGB-stacking results before we note that the bottom adapters (plotted in Green) with very few parameters (around a few thousand) can lead to substantially better results than simply using fixed pretrained models. This holds when bottom adapters are combined with top adapters and even in the absence of top adapters. Although, as noted before the overall results are much poorer without top adapters. From Figure 9.16 we see that bottom adapters help for both NFNet (green bar in column 1, row 1 and column 1, row2) and ResNet (green bar in column 1, row 1 and column 1, row2). Thus, broadly similar results hold across environment suites.

Middle Adapters: From Figure 9.16 also shows that while bottom and top adapters *together* (green bar on the right plots) can achieve good performance there still exists a significant gap compared to the full fine-tuning approach. However, inserting middle adapters, either alone (shown by orange) or together with bottom adapters (shown in purple) leads to a much more improved performance. Overall, using adapters in all the layers is closely able to match the full fine-tuning performance. This substantial effect of middle adapters is not unexpected since the middle part of the network contains a large part of the pretrained network and thus has significant affect on the output representation.

Overall our results show the importance of adding adapters in each of the network section. As noted previously, this usage of adapters in different network sections is different from prior works [126, 184] which only focus on middle adapters. While such middle adapters are sufficient for semantic classification tasks (considered in [126, 184], they are insufficient for control tasks considered here.

9.4.4 Discussion

In this section we briefly discuss some observations on the use of adapters and future works that adapters can enable for robot control tasks.

Sim2Real: Prior work in robotics often use extensive visual domain randomization, photo-realistic simulators, and other sensory modalities such as depth (which have a smaller sim2real gap) to transfer simulation trained policies to real-world. Our results show that using large pre-trained models (which are closer to natural image statistics) can avoid the need for such expensive sim2real strategy. While there still exists a performance gap ($\sim 24\%$ without visual domain randomization, $\sim 53\%$ with domain randomization), we believe this opens up significant avenues for future research.

CLIP/ALIGN Representations: Recent works [158, 180] propose representations that outperform off-the-shelf CLIP representations. However, CLIP representations are much more semantically powerful [51] and highly robust across a range of data distributions [249]. Our results in Section 6.5.2 shows that while off-the-shelf CLIP representations can be poor (especially for RGB-stacking see Figure 6.6 Right), adapting them through our proposed adapters results in similar performance as other adapted representations (such as MAE ones). Moreover, adapting CLIP representations significantly outperforms all fixed off-the-shelf representations. This is important since CLIP representations are much more semantically powerful and have been used for vastly different task – dialogue generation [4], robot navigation [51], images sketching [245]. Our result shows that the same pretrained model can perform robot manipulation and reach close to optimal performance. This provides exciting avenues for future research wherein a single fixed model can be used to solve a very wide range of tasks.

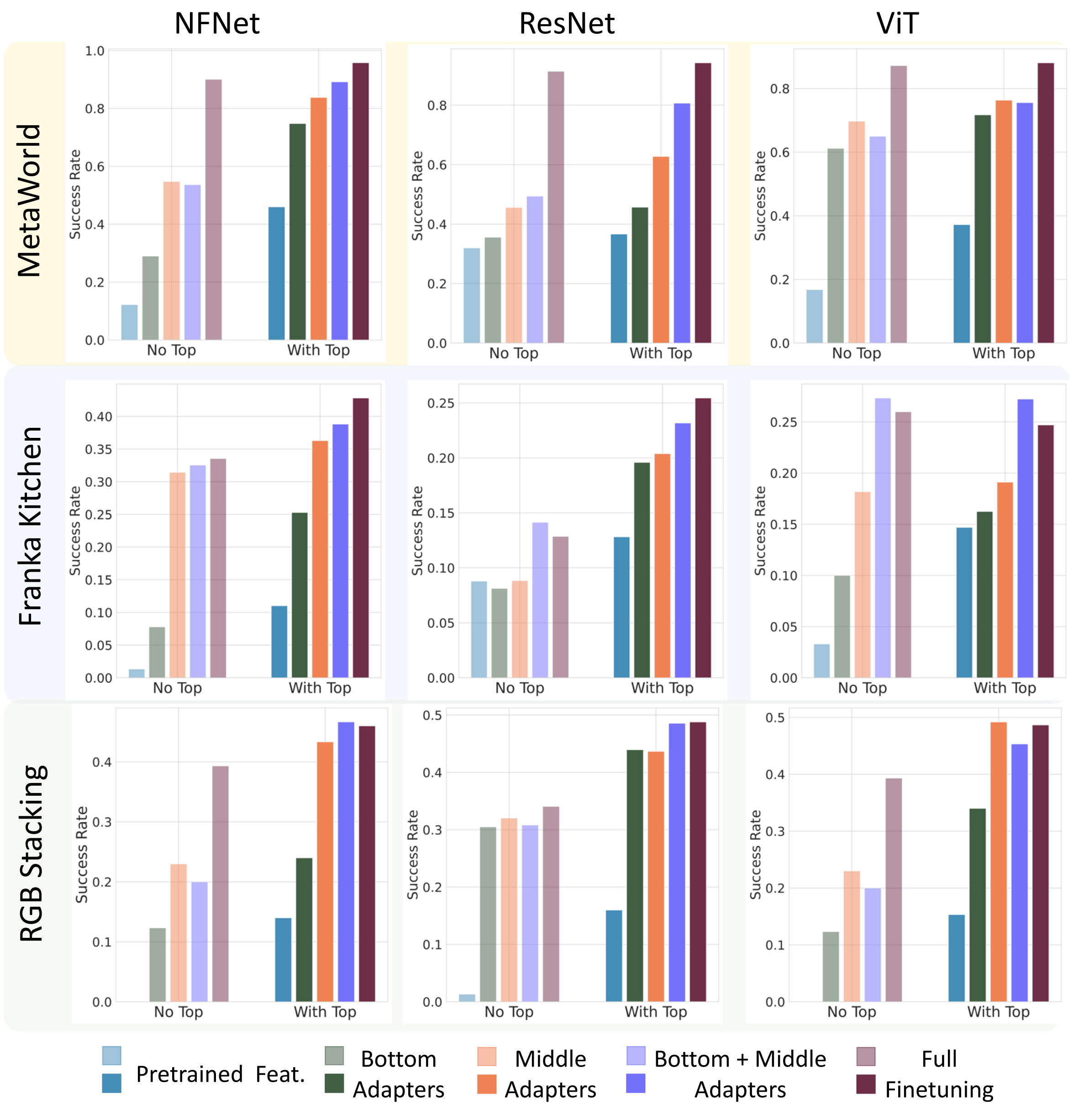


Figure 9.16: Results on the RGB-Stacking environment for 3 different type of model architectures.

Bibliography

1. F. J. Abu-Dakka, B. Nemec, J. A. Jørgensen, T. R. Savarimuthu, N. Krüger, and A. Ude. "Adaptation of manipulation skills in physical contact with the environment to reference force profiles". *Autonomous Robots* 39:2, 2015, pp. 199–217.
2. M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng. "Do As I Can and Not As I Say: Grounding Language in Robotic Affordances". In: *arXiv preprint arXiv:2204.01691*. 2022.
3. E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. "Learning the semantics of object–action relations by observation". *The International Journal of Robotics Research* 30:10, 2011, pp. 1229–1249.
4. J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds, et al. "Flamingo: a visual language model for few-shot learning". *arXiv preprint arXiv:2204.14198*, 2022.
5. R. Aljundi, P. Chakravarty, and T. Tuytelaars. "Expert gate: Lifelong learning with a network of experts". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 3366–3375.
6. B. Ames, A. Thackston, and G. Konidaris. "Learning symbolic representations for planning with parameterized skills". *2018 IEEE International Conference on Intelligent Robots and Systems*, 2018.
7. Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba. "Cross-modal scene networks". *IEEE transactions on pattern analysis and machine intelligence* 40:10, 2017, pp. 2303–2314.
8. J. L. Ba, J. R. Kiros, and G. E. Hinton. "Layer normalization". *arXiv preprint arXiv:1607.06450*, 2016.
9. P.-L. Bacon, J. Harb, and D. Precup. "The option-critic architecture". In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.
10. A. Bagaria, J. Crowley, J. W. N. Lim, and G. Konidaris. "Skill Discovery for Exploration and Planning using Deep Skill Graphs", 2020.
11. D. H. Ballard. "Task Frames in Robot Manipulation." In: *AAAI*. Vol. 19. 1984, p. 109.
12. D. H. Ballard and L. Hartman. "Task frames: Primitives for sensory-motor coordination". *Computer Vision, Graphics, and Image Processing* 36:2-3, 1986, pp. 274–297.
13. P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. "Relational inductive biases, deep learning, and graph networks". *arXiv preprint arXiv:1806.01261*, 2018.

14. P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu. "Interaction networks for learning about objects, relations and physics". *Advances in Neural Information Processing Systems*, 2016.
15. M. Beetz, L. Mösenlechner, and M. Tenorth. "CRAM—A Cognitive Robot Abstract Machine for everyday manipulation in human environments". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 1012–1017.
16. C. C. Beltran-Hernandez, D. Petit, I. G. Ramirez-Alpizar, T. Nishi, S. Kikuchi, T. Matsubara, and K. Harada. "Learning force control for contact-rich manipulation tasks with rigid position-controlled robots". *IEEE Robotics and Automation Letters* 5:4, 2020, pp. 5709–5716.
17. D. Berenson, S. Srinivasa, and J. Kuffner. "Task space regions: A framework for pose-constrained manipulation planning". *The International Journal of Robotics Research* 30:12, 2011, pp. 1435–1460.
18. M. Bogdanovic, M. Khadiv, and L. Righetti. "Learning Variable Impedance Control for Contact Sensitive Tasks". *arXiv preprint arXiv:1907.07500*, 2019.
19. A. Brock, S. De, S. L. Smith, and K. Simonyan. "High-performance large-scale image recognition without normalization". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 1059–1071.
20. A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, et al. "Rt-1: Robotics transformer for real-world control at scale". *arXiv preprint arXiv:2212.06817*, 2022.
21. H. Bruyninckx and J. De Schutter. "Specification of force-controlled actions in the" task frame formalism"-a synthesis". *IEEE transactions on robotics and automation* 12:4, 1996, pp. 581–589.
22. J. Butzke, K. Sapkota, K. Prasad, B. MacAllister, and M. Likhachev. "State lattice with controllers: Augmenting lattice-based path planning with controller-based motion primitives". *IEEE International Conference on Intelligent Robots and Systems*, 2014.
23. R. Calandra, A. Owens, D. Jayaraman, J. Lin, W. Yuan, J. Malik, E. H. Adelson, and S. Levine. "More than a feeling: Learning to grasp and regrasp using vision and touch". *IEEE Robotics and Automation Letters* 3:4, 2018, pp. 3300–3307.
24. S. Calinon. "A tutorial on task-parameterized movement learning and retrieval". *Intelligent service robotics* 9:1, 2016, pp. 1–29.
25. S. Calinon, D. Bruno, and D. G. Caldwell. "A task-parameterized probabilistic model with minimal intervention control". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3339–3344.
26. N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. "End-to-end object detection with transformers". In: *European conference on computer vision*. Springer. 2020, pp. 213–229.
27. L. Castrejon, Y. Aytar, C. Vondrick, H. Pirsiavash, and A. Torralba. "Learning aligned cross-modal representations from weakly aligned data". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2940–2949.
28. A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. "Shapenet: An information-rich 3d model repository". *arXiv preprint arXiv:1512.03012*, 2015.
29. T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. "A simple framework for contrastive learning of visual representations". In: *International conference on machine learning*. PMLR. 2020, pp. 1597–1607.

30. G. Comanici and D. Precup. “Optimal policy switching algorithms for reinforcement learning”. In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. 2010, pp. 709–714.
31. A. Conkey and T. Hermans. “Learning Task Constraints from Demonstration for Hybrid Force/Position Control”. In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2019, pp. 162–169.
32. A. Conkey and T. Hermans. “Planning under Uncertainty to Goal Distributions”. *arXiv preprint arXiv:2011.04782*, 2020.
33. E. Coumans et al. “Bullet physics library”. *Open source: bulletphysics. org* 15:49, 2013, p. 5.
34. B. Curless and M. Levoy. “A volumetric method for building complex models from range images”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 303–312.
35. C. Daniel, H. Van Hoof, J. Peters, and G. Neumann. “Probabilistic inference for determining options in reinforcement learning”. *Machine Learning* 104:2-3, 2016, pp. 337–357.
36. J. De Schutter and H. Van Brussel. “Compliant robot motion I. A formalism for specifying compliant motion tasks”. *The International Journal of Robotics Research* 7:4, 1988, pp. 3–17.
37. H. De Vries, F. Strub, J. Mary, H. Larochelle, O. Pietquin, and A. C. Courville. “Modulating early visual processing by language”. *Advances in Neural Information Processing Systems* 30, 2017.
38. M. Deitke, R. Liu, M. Wallingford, H. Ngo, O. Michel, A. Kusupati, A. Fan, C. Laforte, V. Voleti, S. Y. Gadre, et al. “Objaverse-xl: A universe of 10m+ 3d objects”. *arXiv preprint arXiv:2307.05663*, 2023.
39. B. Deng, K. Genova, S. Yazdani, S. Bouaziz, G. Hinton, and A. Tagliasacchi. “Cvxnet: Learnable convex decomposition”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 31–44.
40. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. “Bert: Pre-training of deep bidirectional transformers for language understanding”. *arXiv preprint arXiv:1810.04805*, 2018.
41. A. Dietrich, C. Ott, and A. Albu-Schäffer. “An overview of null space projections for redundant, torque-controlled robots”. *The International Journal of Robotics Research*, 2015.
42. A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. *arXiv preprint arXiv:2010.11929*, 2020.
43. D. Dwibedi, J. Tompson, C. Lynch, and P. Sermanet. “Learning actionable representations from visual observations”. In: *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2018, pp. 1577–1584.
44. M. Eppe, P. D. Nguyen, and S. Wermter. “From semantics to execution: Integrating action planning with reinforcement learning for robotic causal problem-solving”. *Frontiers in Robotics and AI* 6, 2019, p. 123.
45. N. Fazeli, M. Oller, J. Wu, Z. Wu, J. B. Tenenbaum, and A. Rodriguez. “See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion”. *Science Robotics* 4:26, 2019, eaav3123.
46. M. Fey and J. E. Lenssen. “Fast Graph Representation Learning with PyTorch Geometric”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019.
47. S. Fichtl, A. McManus, W. Mustafa, D. Kraft, N. Krüger, and F. Guerin. “Learning spatial relationships from 3D vision using histograms”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 501–508.

48. P. Florence, L. Manuelli, and R. Tedrake. "Self-supervised correspondence in visuomotor policy learning". *IEEE Robotics and Automation Letters* 5:2, 2019, pp. 492–499.
49. P. R. Florence, L. Manuelli, and R. Tedrake. "Dense object nets: Learning dense visual object descriptors by and for robotic manipulation". *arXiv preprint arXiv:1806.08756*, 2018.
50. K.-i. Funahashi and Y. Nakamura. "Approximation of dynamical systems by continuous time recurrent neural networks". *Neural networks* 6:6, 1993, pp. 801–806.
51. S. Y. Gadre, M. Wortsman, G. Ilharco, L. Schmidt, and S. Song. "CLIP on Wheels: Zero-Shot Object Navigation as Object Localization and Exploration". *arXiv preprint arXiv:2203.10421*, 2022.
52. W. Gao and R. Tedrake. "kPAM 2.0: Feedback Control for Category-Level Robotic Manipulation". *IEEE Robotics and Automation Letters* 6:2, 2021, pp. 2962–2969.
53. C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. "Integrated task and motion planning". *Annual review of control, robotics, and autonomous systems* 4, 2021, pp. 265–293.
54. C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling. "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 30. 2020, pp. 440–448.
55. J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, et al. "Bootstrap your own latent-a new approach to self-supervised learning". *Advances in neural information processing systems* 33, 2020, pp. 21271–21284.
56. A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. "Relay Policy Learning: Solving Long Horizon Tasks via Imitation and Reinforcement Learning". *Conference on Robot Learning (CoRL)*, 2019.
57. T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.
58. R. Hadsell, S. Chopra, and Y. LeCun. "Dimensionality reduction by learning an invariant mapping". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1735–1742.
59. K. Hara, H. Kataoka, and Y. Satoh. "Can Spatiotemporal 3D CNNs Retrace the History of 2D CNNs and ImageNet?" In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018, pp. 6546–6555.
60. M. Hausknecht and P. Stone. "Deep reinforcement learning in parameterized action space". *International Conference on Learning Representations (ICLR)*, 2015.
61. K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. "Masked autoencoders are scalable vision learners". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 16000–16009.
62. K. He, G. Gkioxari, P. Dollár, and R. Girshick. "Mask R-CNN". In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2017.
63. K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
64. D. Henrich, T. Ogasawara, and H. Worn. "Manipulating deformable linear objects-contact states and point contacts". In: *Proceedings of the 1999 IEEE International Symposium on Assembly and Task Planning (ISATP'99)(Cat. No. 99TH8470)*. IEEE. 1999, pp. 198–204.

65. A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. *Stable Baselines*. https://github.com/hill-a/stable-baselines. 2018.
66. A. Hinrichs, D. Krieg, R. J. Kunsch, and D. Rudolf. "Expected dispersion of uniformly distributed points". *Journal of Complexity* 61, 2020, p. 101483.
67. N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. "Parameter-efficient transfer learning for NLP". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2790–2799.
68. Y. Huang, J. Silvério, L. Rozo, and D. G. Caldwell. "Generalized task-parameterized skill learning". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 5667–5474.
69. S. Hutchinson, G. D. Hager, and P. I. Corke. "A tutorial on visual servo control". *IEEE transactions on robotics and automation* 12:5, 1996, pp. 651–670.
70. B. Ichter, P. Sermanet, and C. Lynch. "Broadly-Exploring, Local-Policy Trees for Long-Horizon Task Planning". *arXiv preprint arXiv:2010.06491*, 2020.
71. A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal. "Dynamical movement primitives: learning attractor models for motor behaviors". *Neural computation* 25:2, 2013, pp. 328–373.
72. S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. "Rlbench: The robot learning benchmark & learning environment". *IEEE Robotics and Automation Letters* 5:2, 2020, pp. 3019–3026.
73. E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. "Bc-z: Zero-shot task generalization with robotic imitation learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 991–1002.
74. M. Janner, S. Levine, W. T. Freeman, J. B. Tenenbaum, C. Finn, and J. Wu. "Reasoning about physical interactions with object-oriented prediction and planning". *International Conference on Learning Representations (ICLR)*, 2019.
75. M. Janner, I. Mordatch, and S. Levine. "gamma-Models: Generative Temporal Difference Learning for Infinite-Horizon Prediction". *Advances in Neural Information Processing Systems*, 2020.
76. R. Jeong, Y. Aytar, D. Khosid, Y. Zhou, J. Kay, T. Lampe, K. Bousmalis, and F. Nori. "Self-supervised sim-to-real adaptation for visual robotic manipulation". In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2020, pp. 2718–2724.
77. N. Jetchev, T. Lang, and M. Toussaint. "Learning grounded relational symbols from continuous data for abstract reasoning", 2013.
78. D. Ji and M. A. Wilson. "Coordinated memory replay in the visual cortex and hippocampus during sleep". *Nature neuroscience* 10:1, 2007, pp. 100–107.
79. C. Jia, Y. Yang, Y. Xia, Y.-T. Chen, Z. Parekh, H. Pham, Q. Le, Y.-H. Sung, Z. Li, and T. Duerig. "Scaling up visual and vision-language representation learning with noisy text supervision". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 4904–4916.
80. M. Jia, L. Tang, B.-C. Chen, C. Cardie, S. Belongie, B. Hariharan, and S.-N. Lim. "Visual prompt tuning". *arXiv preprint arXiv:2203.12119*, 2022.
81. L. Johannsmeier, M. Gerchow, and S. Haddadin. "A framework for robot manipulation: Skill formalism, meta learning and adaptive control". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 5844–5850.

82. E. Johns. "Coarse-to-fine imitation learning: Robot manipulation from a single demonstration". In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2021, pp. 4613–4619.
83. R. Jonschkowski and O. Brock. "State Representation Learning in Robotics: Using Prior Knowledge about Physical Interaction." In: *Robotics: Science and Systems*. 2014.
84. R. Jonschkowski, R. Hafner, J. Scholz, and M. Riedmiller. "Pves: Position-velocity encoders for unsupervised learning of structured state representations". *arXiv preprint arXiv:1705.09805*, 2017.
85. P. Jund, A. Eitel, N. Abdo, and W. Burgard. "Optimization beyond the convolution: Generalizing spatial relations with end-to-end metric learning". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 1–7.
86. L. P. Kaelbling. "Hierarchical learning in stochastic domains: Preliminary results". In: *Proceedings of the tenth international conference on machine learning*. Vol. 951. 1993, pp. 167–173.
87. L. P. Kaelbling. "Learning to achieve goals". In: *IJCAI*. Vol. 2. Citeseer. 1993, pp. 1094–8.
88. L. P. Kaelbling and T. Lozano-Pérez. "Hierarchical task and motion planning in the now". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1470–1477.
89. L. P. Kaelbling and T. Lozano-Pérez. "Integrated task and motion planning in belief space". *The International Journal of Robotics Research* 32:9-10, 2013, pp. 1194–1227.
90. D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation". *arXiv preprint arXiv:1806.10293*, 2018.
91. A. Kamath, M. Singh, Y. LeCun, G. Synnaeve, I. Misra, and N. Carion. "Mdetr-modulated detection for end-to-end multi-modal understanding". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1780–1790.
92. A. Karami, H. Sadeghian, M. Keshmiri, and G. Oriolo. "Hierarchical tracking task control in redundant manipulators with compliance control in the null-space". *Mechatronics*, 2018.
93. R. Kartmann, F. Paus, M. Grotz, and T. Asfour. "Extraction of physically plausible support relations to predict and validate manipulation action effects". *IEEE Robotics and Automation Letters* 3:4, 2018, pp. 3991–3998.
94. I. Kauvar, C. Doyle, L. Zhou, and N. Haber. "Curious replay for model-based adaptation". *arXiv preprint arXiv:2306.15934*, 2023.
95. A. Khandelwal, L. Weihs, R. Mottaghi, and A. Kembhavi. "Simple but effective: Clip embeddings for embodied ai". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 14829–14838.
96. O. Khatib. "A unified approach for motion and force control of robot manipulators: The operational space formulation". *IEEE Journal on Robotics and Automation* 3:1, 1987, pp. 43–53.
97. P. Kidger, J. Morrill, J. Foster, and T. Lyons. "Neural controlled differential equations for irregular time series". *Advances in Neural Information Processing Systems* 33, 2020, pp. 6696–6707.
98. S.-K. Kim and M. Likhachev. "Parts assembly planning under uncertainty with simulation-aided physical reasoning". In: *2017 IEEE International Conference on Robotics and Automation*. IEEE. 2017, pp. 4074–4081.
99. S.-K. Kim, O. Salzman, and M. Likhachev. "POMHDP: Search-based belief space planning using multiple heuristics". In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 734–744.

100. J. E. King, M. Cognetti, and S. S. Srinivasa. “Rearrangement planning using object-centric and robot-centric action spaces”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3940–3947.
101. D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980*, 2014.
102. T. N. Kipf and M. Welling. “Semi-supervised classification with graph convolutional networks”. *arXiv preprint arXiv:1609.02907*, 2016.
103. J. Kober, M. Gienger, and J. J. Steil. “Learning movement primitives for force interaction tasks”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 3192–3199.
104. G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. “From skills to symbols: Learning symbolic representations for abstract high-level planning”. *Journal of Artificial Intelligence Research* 61, 2018, pp. 215–289.
105. G. D. Konidaris and A. G. Barto. “Efficient Skill Learning using Abstraction Selection.” In: Citeseer. 2009.
106. I. Kostrikov, D. Yarats, and R. Fergus. “Image augmentation is all you need: Regularizing deep reinforcement learning from pixels”. *arXiv preprint arXiv:2004.13649*, 2020.
107. D. Kragic, H. I. Christensen, et al. “Survey on visual servoing for manipulation”. *Computational Vision and Active Perception Laboratory, Fiskartorpsv* 15, 2002, p. 2002.
108. S. Krishnan, R. Fox, I. Stoica, and K. Goldberg. “Ddco: Discovery of deep continuous options for robot learning from demonstrations”. *arXiv preprint arXiv:1710.05421*, 2017.
109. O. Kroemer, S. Niekum, and G. Konidaris. “A review of robot learning for manipulation: Challenges, representations, and algorithms”. *arXiv preprint arXiv:1907.03146*, 2019.
110. O. Kroemer, S. Niekum, and G. D. Konidaris. “A review of robot learning for manipulation: Challenges, representations, and algorithms”. *Journal of machine learning research* 22:30, 2021.
111. O. Kroemer and G. S. Sukhatme. “Learning spatial preconditions of manipulation skills using random forests”. In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 676–683.
112. O. Kroemer, H. Van Hoof, G. Neumann, and J. Peters. “Learning to predict phases of manipulation tasks as hidden states”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 4009–4014.
113. T. Kroger, B. Finkemeyer, and F. M. Wahl. “A task frame formalism for practical implementations”. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA’04. 2004*. Vol. 5. IEEE. 2004, pp. 5218–5223.
114. J. Kulick, M. Toussaint, T. Lang, and M. Lopes. “Active learning for teaching a robot grounded relational symbols”. In: *Twenty-Third International Joint Conference on Artificial Intelligence*. 2013.
115. N. O. Lambert, A. Wilcox, H. Zhang, K. S. Pister, and R. Calandra. “Learning Accurate Long-term Dynamics for Model-based Reinforcement Learning”. *arXiv preprint arXiv:2012.09156*, 2020.
116. M. Laskin, A. Srinivas, and P. Abbeel. “Curl: Contrastive unsupervised representations for reinforcement learning”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5639–5650.
117. M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. “Reinforcement learning with augmented data”. *Advances in neural information processing systems* 33, 2020, pp. 19884–19895.
118. S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

119. A. X. Lee, C. M. Devin, Y. Zhou, T. Lampe, K. Bousmalis, J. T. Springenberg, A. Byravan, A. Abdolmaleki, N. Gileadi, D. Khosid, et al. "Beyond pick-and-place: Tackling robotic stacking of diverse shapes". In: *5th Annual Conference on Robot Learning*. 2021.
120. A. X. Lee, C. Devin, Y. Zhou, T. Lampe, K. Bousmalis, J. T. Springenberg, A. Byravan, A. Abdolmaleki, N. Gileadi, D. Khosid, C. Fantacci, J. E. Chen, A. Raju, R. Jeong, M. Neunert, A. Laurens, S. Saliceti, F. Casarini, M. Riedmiller, R. Hadsell, and F. Nori. "Beyond Pick-and-Place: Tackling Robotic Stacking of Diverse Shapes". In: *Conference on Robot Learning (CoRL)*. 2021. URL: https://openreview.net/forum?id=U0Q8CrtBJxJ.
121. M. A. Lee, Y. Zhu, K. Srinivasan, P. Shah, S. Savarese, L. Fei-Fei, A. Garg, and J. Bohg. "Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 8943–8950.
122. S. Levine, C. Finn, T. Darrell, and P. Abbeel. "End-to-end training of deep visuomotor policies". *The Journal of Machine Learning Research* 17:1, 2016, pp. 1334–1373.
123. J. Li, R. Selvaraju, A. Gotmare, S. Joty, C. Xiong, and S. C. H. Hoi. "Align before fuse: Vision and language representation learning with momentum distillation". *Advances in neural information processing systems* 34, 2021, pp. 9694–9705.
124. Q. Li, O. Kroemer, Z. Su, F. F. Veiga, M. Kaboli, and H. J. Ritter. "A review of tactile information: Perception and action through touch". *IEEE Transactions on Robotics* 36:6, 2020, pp. 1619–1634.
125. T. Li, R. Calandra, D. Pathak, Y. Tian, F. Meier, and A. Rai. "Planning in learned latent action spaces for generalizable legged locomotion". *IEEE Robotics and Automation Letters* 6:2, 2021, pp. 2682–2689.
126. W.-H. Li, X. Liu, and H. Bilen. "Cross-domain Few-shot Learning with Task-specific Adapters". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 7161–7170.
127. X. L. Li and P. Liang. "Prefix-tuning: Optimizing continuous prompts for generation". *arXiv preprint arXiv:2101.00190*, 2021.
128. J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. "Code as policies: Language model programs for embodied control". *arXiv preprint arXiv:2209.07753*, 2022.
129. J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. "GPU-accelerated robotic simulation for distributed reinforcement learning". *Conference on Robot Learning (CoRL)*, 2018.
130. J. Liang, M. Sharma, A. LaGrassa, S. Vats, S. Saxena, and O. Kroemer. "Search-based task planning with learned skill effect models for lifelong robotic manipulation". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 6351–6357.
131. R. Liaw, S. Krishnan, A. Garg, D. Crankshaw, J. E. Gonzalez, and K. Goldberg. *Composing Meta-Policies for Autonomous Driving Using Hierarchical Deep Reinforcement Learning*. 2017. arXiv: 1711.01503 [cs.AI].
132. K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. "Text2motion: From natural language instructions to feasible plans". *arXiv preprint arXiv:2303.12153*, 2023.
133. X. Liu, K. Ji, Y. Fu, Z. Du, Z. Yang, and J. Tang. "P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks". *arXiv preprint arXiv:2110.07602*, 2021.
134. K. Lu, A. Grover, P. Abbeel, and I. Mordatch. "Reset-Free Lifelong Learning with Skill-Space Planning". *International Conference on Learning Representations (ICLR)*, 2021.

135. M. C. Machado, M. G. Bellemare, and M. Bowling. "A laplacian framework for option discovery in reinforcement learning". *arXiv preprint arXiv:1703.00956*, 2017.
136. R. K. Mahabadi, S. Ruder, M. Dehghani, and J. Henderson. "Parameter-efficient multi-task fine-tuning for transformers via shared hypernetworks". *arXiv preprint arXiv:2106.04489*, 2021.
137. A. Mandlekar, F. Ramos, B. Boots, S. Savarese, L. Fei-Fei, A. Garg, and D. Fox. "Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data". In: *2020 IEEE International Conference on Robotics and Automation*. IEEE. 2020, pp. 4414–4420.
138. S. Manschitz, M. Gienger, J. Kober, and J. Peters. "Learning Sequential Force Interaction Skills". *Robotics* 9:2, 2020, p. 45.
139. S. Manschitz, J. Kober, M. Gienger, and J. Peters. "Learning to sequence movement primitives from demonstrations". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 4414–4421.
140. L. Manuelli, W. Gao, P. Florence, and R. Tedrake. "kpam: Keypoint affordances for category-level robotic manipulation". *arXiv preprint arXiv:1903.06684*, 2019.
141. R. Martín-Martín, M. Lee, R. Gardner, S. Savarese, J. Bohg, and A. Garg. "Variable Impedance Control in End-Effector Space. An Action Space for Reinforcement Learning in Contact Rich Tasks". In: *Proceedings of the International Conference of Intelligent Robots and Systems (IROS)*. 2019.
142. M. T. Mason. "Compliance and force control for computer controlled manipulators". *IEEE Transactions on Systems, Man, and Cybernetics* 11:6, 1981, pp. 418–432.
143. W. Masson, P. Ranchod, and G. Konidaris. "Reinforcement learning with parameterized actions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.
144. C. Matuszek, L. Bo, L. Zettlemoyer, and D. Fox. "Learning from unscripted deictic gesture and language for human-robot interactions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 28. 1. 2014.
145. D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. *PDDL-the planning domain definition language*. 1998.
146. A. McGovern and A. G. Barto. "Automatic discovery of subgoals in reinforcement learning using diverse density", 2001.
147. O. Mees, N. Abdo, M. Mazuran, and W. Burgard. "Metric learning for generalizing spatial relations to new objects". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 3175–3182.
148. O. Mees, L. Hermann, and W. Burgard. "What matters in language conditioned robotic imitation learning over unstructured data". *IEEE Robotics and Automation Letters* 7:4, 2022, pp. 11205–11212.
149. O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard. "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks". *IEEE Robotics and Automation Letters* 7:3, 2022, pp. 7327–7334.
150. L. Metz, J. Ibarz, N. Jaitly, and J. Davidson. "Discrete sequential prediction of continuous actions for deep rl". *arXiv preprint arXiv:1705.05035*, 2017.
151. T. Migimatsu and J. Bohg. "Object-Centric Task and Motion Planning in Dynamic Environments". *IEEE Robotics and Automation Letters* 5:2, 2020, pp. 844–851.

152. A. S. Morgan, B. Wen, J. Liang, A. Boularias, A. M. Dollar, and K. Bekris. "Vision-driven compliant manipulation for reliable, high-precision assembly tasks". *arXiv preprint arXiv:2106.14070*, 2021.
153. J. D. Morrow and P. K. Khosla. "Manipulation task primitives for composing robot skills". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. 1997, 3354–3359 vol.4. DOI: `10.1109/ROBOT.1997.606800`.
154. C. Mucchiani and M. Yim. "Dynamic grasping for object picking using passive zero-dof end-effectors". *IEEE Robotics and Automation Letters* 6:2, 2021, pp. 3089–3096.
155. M. Mühlig, M. Gienger, J. J. Steil, and C. Goerick. "Automatic selection of task spaces for imitation learning". In: *International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 4996–5002.
156. P. Naderian, G. Loaiza-Ganem, H. J. Braviner, A. L. Caterini, J. C. Cresswell, T. Li, and A. Garg. "C-Learning: Horizon-Aware Cumulative Accessibility Estimation". *International Conference on Learning Representations (ICLR)*, 2021.
157. U. Nagarajan, G. Kantor, and R. Hollis. "The ballbot: An omnidirectional balancing mobile robot". *The International Journal of Robotics Research* 33:6, 2014, pp. 917–930.
158. S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. "R3m: A universal visual representation for robot manipulation". *arXiv preprint arXiv:2203.12601*, 2022.
159. Y. Nakamura, H. Hanafusa, and T. Yoshikawa. "Task-priority based redundancy control of robot manipulators". *The International Journal of Robotics Research* 6:2, 1987, pp. 3–15.
160. T. Narita and O. Kroemer. "Policy blending and recombination for multimodal contact-rich tasks". *IEEE Robotics and Automation Letters* 6:2, 2021, pp. 2721–2728.
161. S. Nasiriany, V. H. Pong, S. Lin, and S. Levine. "Planning with goal-conditioned policies". *Advances in Neural Information Processing Systems*, 2019.
162. S. Nasiriany, V. H. Pong, A. Nair, A. Khazatsky, G. Berseth, and S. Levine. "Disco rl: Distribution-conditioned reinforcement learning for general-purpose policies". *International Conference on Robotics and Automation*, 2021.
163. B. J. Nelson, J. D. Morrow, and P. K. Khosla. "Improved force control through visual servoing". In: *Proceedings of 1995 American Control Conference-ACC'95*. Vol. 1. IEEE. 1995, pp. 380–386.
164. Nvidia. *Isaac Gym*. URL: `developer.nvidia.com/isaac-gym`.
165. H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. "Deep metric learning via lifted structured feature embedding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 4004–4012.
166. V. Pacelli and A. Majumdar. "Learning task-driven control policies via information bottlenecks". *arXiv preprint arXiv:2002.01428*, 2020.
167. A. Padalkar, A. Pooley, A. Jain, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Singh, A. Brohan, et al. "Open x-embodiment: Robotic learning datasets and rt-x models". *arXiv preprint arXiv:2310.08864*, 2023.
168. Z. Pan and K. Hauser. "Decision Making in Joint Push-Grasp Action Space for Large-Scale Object Sorting". *International Conference on Robotics and Automation*, 2020.
169. A. Paraschos, C. Daniel, J. Peters, and G. Neumann. "Using probabilistic movement primitives in robotics". *Autonomous Robots* 42:3, 2018, pp. 529–551.
170. J. Pari, N. M. Shafiullah, S. P. Arunachalam, and L. Pinto. "The surprising effectiveness of representation learning for visual imitation". *arXiv preprint arXiv:2112.01511*, 2021.

171. S. Parisi, A. Rajeswaran, S. Purushwalkam, and A. Gupta. "The unsurprising effectiveness of pre-trained vision models for control". *arXiv preprint arXiv:2203.03580*, 2022.
172. H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. "Learning symbolic models of stochastic domains". *Journal of Artificial Intelligence Research* 29, 2007, pp. 309–352.
173. D. Pathak, D. Gandhi, and A. Gupta. "Self-Supervised Exploration via Disagreement". In: *ICML*. 2019.
174. C. Paxton, F. Jonathan, M. Kobilarov, and G. D. Hager. "Do what i want, not what i did: Imitation of skills by planning sequences of actions". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 3778–3785.
175. J. Pazis and M. G. Lagoudakis. "Reinforcement learning in multidimensional continuous action spaces". In: *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE. 2011, pp. 97–104.
176. E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville. "Film: Visual reasoning with a general conditioning layer". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.
177. L. Peternel, L. Rozo, D. Caldwell, and A. Ajoudani. "A method for derivation of robot task-frame control authority from repeated sensory observations". *IEEE Robotics and Automation Letters* 2:2, 2017, pp. 719–726.
178. M. Q. Pham, J.-M. Crego, F. Yvon, and J. Senellart. "A study of residual adapters for multi-domain neural machine translation". In: *Conference on Machine Translation*. 2020.
179. A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 8748–8763.
180. I. Radosavovic, T. Xiao, S. James, P. Abbeel, J. Malik, and T. Darrell. "Real-World Robot Learning with Masked Visual Pre-training". *arXiv preprint arXiv:2210.03109*, 2022.
181. M. H. Raibert and J. J. Craig. "Hybrid position/force control of manipulators", 1981.
182. A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen. "Hierarchical text-conditional image generation with clip latents". *arXiv preprint arXiv:2204.06125*, 2022.
183. A. Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars. "Encoder based lifelong learning". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 1320–1328.
184. S.-A. Rebuffi, H. Bilen, and A. Vedaldi. "Learning multiple visual domains with residual adapters". *Advances in neural information processing systems* 30, 2017.
185. S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. "A generalist agent". *arXiv preprint arXiv:2205.06175*, 2022.
186. J. Reizenstein, R. Shapovalov, P. Henzler, L. Sbordone, P. Labatut, and D. Novotny. "Common objects in 3d: Large-scale learning and evaluation of real-life 3d category reconstruction". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10901–10911.
187. M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. Wiele, V. Mnih, N. Heess, and J. T. Springenberg. "Learning by playing solving sparse reward tasks from scratch". In: *International conference on machine learning*. PMLR. 2018, pp. 4344–4353.
188. E. Rohmer, S. P. Singh, and M. Freese. "V-REP: A versatile and scalable robot simulation framework". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1321–1326.

189. A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone. "3D dynamic scene graphs: Actionable spatial perception with places, objects, and humans". *arXiv preprint arXiv:2002.06289*, 2020.
190. B. Rosman and S. Ramamoorthy. "Learning spatial relationships between objects". *The International Journal of Robotics Research* 30:11, 2011, pp. 1328–1342.
191. B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, et al. "Code llama: Open foundation models for code". *arXiv preprint arXiv:2308.12950*, 2023.
192. A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. "Progressive neural networks". *arXiv preprint arXiv:1606.04671*, 2016.
193. C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, et al. "Photorealistic Text-to-Image diffusion models with deep language understanding (2022)". *URL https://arxiv. org/abs/2205.11487*.
194. A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. "A simple neural network module for relational reasoning". In: *Advances in neural information processing systems*. 2017, pp. 4967–4976.
195. S. Saxena, A. LaGrassa, and O. Kroemer. "Learning reactive and predictive differentiable controllers for switching linear dynamical models". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7563–7569.
196. S. Saxena, M. Sharma, and O. Kroemer. "Multi-Resolution Sensing for Real-Time Control with Vision-Language Models". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2210–2228.
197. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The graph neural network model". *IEEE Transactions on Neural Networks* 20:1, 2008, pp. 61–80.
198. T. Schaul, D. Horgan, K. Gregor, and D. Silver. "Universal value function approximators". In: *International conference on machine learning*. PMLR. 2015, pp. 1312–1320.
199. T. Schmidt, R. Newcombe, and D. Fox. "Self-supervised visual descriptor learning for dense correspondence". *IEEE Robotics and Automation Letters* 2:2, 2016, pp. 420–427.
200. F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 815–823.
201. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347*, 2017.
202. J. Schwarz, W. Czarnecki, J. Luketina, A. Grabska-Barwinska, Y. W. Teh, R. Pascanu, and R. Hadsell. "Progress & compress: A scalable framework for continual learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4528–4537.
203. L. Sciavicco and B. Siciliano. *Modelling and control of robot manipulators*. Springer Science & Business Media, 2012.
204. M. Y. Seker, A. E. Tekden, and E. Ugur. "Deep effect trajectory prediction in robot manipulation". *Robotics and Autonomous Systems* 119, 2019, pp. 173–184.
205. R. Shah and V. Kumar. "RRL: Resnet as representation for Reinforcement Learning". In: *International Conference on Machine Learning*. PMLR. 2021.
206. A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. "Dynamics-aware unsupervised discovery of skills". *International Conference on Learning Representations (ICLR)*, 2019.
207. M. Sharma, C. Fantacci, Y. Zhou, S. Koppula, N. Heess, J. Scholz, and Y. Aytar. "Lossless Adaptation of Pretrained Vision Models For Robotic Manipulation". In: *The Eleventh International Conference on Learning Representations*.

208. M. Sharma, C. Fantacci, Y. Zhou, S. Koppula, N. Heess, J. Scholz, and Y. Aytar. “Lossless adaptation of pretrained vision models for robotic manipulation”. *arXiv preprint arXiv:2304.06600*, 2023.
209. M. Sharma and O. Kroemer. “Efficiently Learning Manipulations by Selecting Structured Skill Representations”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 1039–1046.
210. M. Sharma and O. Kroemer. “Generalizing Object-Centric Task-Axes Controllers using Keypoints”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7548–7554.
211. M. Sharma and O. Kroemer. “Relational Learning for Skill Preconditions”. In: *Conference on Robot Learning*. PMLR. 2021, pp. 845–861.
212. M. Sharma and O. Kroemer. “Relational learning for skill preconditions”. *arXiv preprint arXiv:2012.01693*, 2020.
213. M. Sharma, J. Liang, J. Zhao, A. LaGrassa, and O. Kroemer. “Learning to compose hierarchical object-centric controllers for robotic manipulation”. *arXiv preprint arXiv:2011.04627*, 2020.
214. M. Sharma, A. Sharma, N. Rhinehart, and K. M. Kitani. “Directed-Info GAIL: Learning Hierarchical Policies from Unsegmented Demonstrations using Directed Information”. In: *International Conference on Learning Representations*. 2019.
215. R. N. Shepard. “Perceptual-cognitive universals as reflections of the world”. *Psychonomic Bulletin & Review* 1:1, 1994, pp. 2–28.
216. J. Shi, J. Z. Woodruff, P. B. Umbanhowar, and K. M. Lynch. “Dynamic in-hand sliding manipulation”. *IEEE Transactions on Robotics* 33:4, 2017, pp. 778–795.
217. K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner. “Taco: Learning task decomposition via temporal alignment for control”. *arXiv preprint arXiv:1803.01840*, 2018.
218. H. Shin, J. K. Lee, J. Kim, and J. Kim. “Continual learning with deep generative replay”. *Advances in neural information processing systems* 30, 2017.
219. M. Shridhar, L. Manuelli, and D. Fox. “Cliport: What and where pathways for robotic manipulation”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 894–906.
220. M. Shridhar, L. Manuelli, and D. Fox. “Perceiver-actor: A multi-task transformer for robotic manipulation”. *arXiv preprint arXiv:2209.05451*, 2022.
221. R. Shu and R. Hollis. “Momentum based Whole-Body Optimal Planning for a Single-Spherical-Wheeled Balancing Mobile Manipulator”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 3221–3226.
222. A. Simeonov, Y. Du, B. Kim, F. R. Hogan, J. Tenenbaum, P. Agrawal, and A. Rodriguez. “A Long Horizon Planning Framework for Manipulating Rigid Pointcloud Objects”. *Conference on Robot Learning (CoRL)*, 2020.
223. K. Simonyan and A. Zisserman. “Very deep convolutional networks for large-scale image recognition”. *arXiv preprint arXiv:1409.1556*, 2014.
224. Ö. Şimşek, A. P. Wolfe, and A. G. Barto. “Identifying useful subgoals in reinforcement learning by local graph partitioning”. In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 816–823.
225. A. Singh, R. Hu, V. Goswami, G. Couairon, W. Galuba, M. Rohrbach, and D. Kiela. “Flava: A foundational language and vision alignment model”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 15638–15650.

226. I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. "Progprompt: Generating situated robot task plans using large language models". *arXiv preprint arXiv:2209.11302*, 2022.
227. S. Singh, F. M. Ramirez, J. Varley, A. Zeng, and V. Sindhwani. "Multiscale sensor fusion and continuous control with neural CDEs". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 10897–10904.
228. L. Smith and M. Gasser. "The development of embodied cognition: Six lessons from babies". *Artificial life* 11:1-2, 2005, pp. 13–29.
229. N. Somavarapu, C.-Y. Ma, and Z. Kira. "Frustratingly simple domain generalization via image stylization". *arXiv preprint arXiv:2006.11207*, 2020.
230. H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork. "Multi-Object Rearrangement with Monte Carlo Tree Search: A Case Study on Planar Nonprehensile Sorting". *International Conference on Intelligent Robots and Systems (IROS)*, 2019.
231. O. Spector and D. Di Castro. "Insertionnet-a scalable solution for insertion". *IEEE Robotics and Automation Letters* 6:3, 2021, pp. 5509–5516.
232. A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn. "Universal planning networks: Learning generalizable representations for visuomotor control". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 4732–4741.
233. M. Stolle and D. Precup. "Learning options in reinforcement learning". In: *International Symposium on abstraction, reformulation, and approximation*. Springer. 2002, pp. 212–223.
234. A. Stone, T. Xiao, Y. Lu, K. Gopalakrishnan, K.-H. Lee, Q. Vuong, P. Wohlhart, B. Zitkovich, F. Xia, C. Finn, and K. Hausman. "Open-World Object Manipulation using Pre-Trained Vision-Language Model". In: *arXiv preprint*. 2023.
235. A. Suárez-Hernández, T. Gaugry, J. Segovia-Aguas, A. Bernardin, C. Torras, M. Marchal, and G. Alenyà. "Leveraging Multiple Environments for Learning and Decision Making: a Dismantling Use Case". *IEEE International Conference on Intelligent Robots and Systems*, 2020.
236. Y.-L. Sung, J. Cho, and M. Bansal. "Lst: Ladder side-tuning for parameter and memory efficient transfer learning". *arXiv preprint arXiv:2206.06522*, 2022.
237. M. Suomalainen, Y. Karayiannidis, and V. Kyrki. "A Survey of Robot Manipulation in Contact". *arXiv preprint arXiv:2112.01942*, 2021.
238. R. S. Sutton, D. Precup, and S. Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". *Artificial intelligence* 112:1-2, 1999, pp. 181–211.
239. A. E. Tekden, A. Erdem, E. Erdem, T. Asfour, and E. Ugur. "Object and Relation Centric Representations for Push Effect Prediction". *arXiv preprint arXiv:2102.02100*, 2021.
240. S. Tellex, T. Kollar, S. Dickerson, M. Walter, A. Banerjee, S. Teller, and N. Roy. "Understanding natural language commands for robotic navigation and mobile manipulation". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 25. 1. 2011, pp. 1507–1514.
241. S. Thrun and T. M. Mitchell. "Lifelong robot learning". *Robotics and autonomous systems* 15:1-2, 1995, pp. 25–46.
242. E. Ugur and J. Piater. "Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning". In: *2015 IEEE International Conference on Robotics and Automation*. IEEE. 2015, pp. 2627–2633.
243. A. L. P. Ureche, K. Umezawa, Y. Nakamura, and A. Billard. "Task parameterization using continuous constraints extracted from human demonstrations". *IEEE Transactions on Robotics* 31:6, 2015, pp. 1458–1471.

244. M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller. "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards". *arXiv preprint arXiv:1707.08817*, 2017.
245. Y. Vinker, E. Pajouheshgar, J. Y. Bo, R. C. Bachmann, A. H. Bermano, D. Cohen-Or, A. Zamir, and A. Shamir. *CLIPasso: Semantically-Aware Object Sketching*. 2022. arXiv: `2202.05822 [cs.GR]`.
246. M. R. Walter, S. M. Hemachandra, B. S. Homberg, S. Tellex, and S. Teller. "Learning semantic maps from natural language descriptions". In: Robotics: Science and Systems. 2013.
247. Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Perez. "Learning compositional models of robot skills for task and motion planning". *The International Journal of Robotics Research* 40:6-7, 2021, pp. 866–894.
248. M. A. Wilson and B. L. McNaughton. "Reactivation of hippocampal ensemble memories during sleep". *Science* 265:5172, 1994, pp. 676–679.
249. M. Wortsman, G. Ilharco, J. W. Kim, M. Li, S. Kornblith, R. Roelofs, R. G. Lopes, H. Hajishirzi, A. Farhadi, H. Namkoong, et al. "Robust fine-tuning of zero-shot models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 7959–7971.
250. T. Xiao, H. Chan, P. Sermanet, A. Wahid, A. Brohan, K. Hausman, S. Levine, and J. Tompson. "Robotic Skill Acquisition via Instruction Augmentation with Vision-Language Models". *arXiv preprint arXiv:2211.11736*, 2022.
251. T. Xiao, I. Radosavovic, T. Darrell, and J. Malik. "Masked visual pre-training for motor control". *arXiv preprint arXiv:2203.06173*, 2022.
252. K. Xie, H. Bharadhwaj, D. Hafner, A. Garg, and F. Shkurti. "Skill Transfer via Partially Amortized Hierarchical Planning". *International Conference on Learning Representations (ICLR)*, 2021.
253. J. Xiong, Q. Wang, Z. Yang, P. Sun, L. Han, Y. Zheng, H. Fu, T. Zhang, J. Liu, and H. Liu. "Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space". *arXiv preprint arXiv:1810.06394*, 2018.
254. D. Xu, A. Mandlekar, R. Martın-Martın, Y. Zhu, S. Savarese, and L. Fei-Fei. "Deep Affordance Foresight: Planning Through What Can Be Done in the Future". *arXiv preprint arXiv:2011.08424*, 2020.
255. A. Yamaguchi and C. G. Atkeson. "Combining finger vision and optical tactile sensing: Reducing and handling errors while cutting vegetables". In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 1045–1051.
256. J. Yoon, E. Yang, J. Lee, and S. J. Hwang. "Lifelong learning with dynamically expandable networks". *arXiv preprint arXiv:1708.01547*, 2017.
257. B. H. Yoshimi and P. K. Allen. "Active, uncalibrated visual servoing". In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE. 1994, pp. 156–161.
258. T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning". In: *Conference on robot learning*. PMLR. 2020, pp. 1094–1100.
259. L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, C. Li, et al. "Florence: A new foundation model for computer vision". *arXiv preprint arXiv:2111.11432*, 2021.
260. W. Yuan, S. Dong, and E. H. Adelson. "Gelsight: High-resolution robot tactile sensors for estimating geometry and force". *Sensors* 17:12, 2017, p. 2762.
261. K. Zampogiannis, Y. Yang, C. Fermüller, and Y. Aloimonos. "Learning the spatial semantics of manipulation actions through preposition grounding". In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 1389–1396.

262. M. D. Zeiler and R. Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.
263. K. Zhang, M. Sharma, J. Liang, and O. Kroemer. “A modular robotic arm control stack for research: Franka-interface and frankapy”. *arXiv preprint arXiv:2011.02398*, 2020.
264. K. Zhang, M. Sharma, M. Veloso, and O. Kroemer. “Leveraging multimodal haptic sensory data for robust cutting”. In: *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2019, pp. 409–416.
265. R. Zhang, Y. Zheng, X. Mao, and M. Huang. “Unsupervised Domain Adaptation with Adapter”. *arXiv preprint arXiv:2111.00667*, 2021.
266. K. Zheng, X. Chen, O. C. Jenkins, and X. E. Wang. “Vlmbench: A compositional benchmark for vision-and-language manipulation”. *arXiv preprint arXiv:2206.08522*, 2022.
267. Y. Zhou, S. Sonawani, M. Phielipp, S. Stepputtis, and H. B. Amor. “Modularity through Attention: Efficient Training and Transfer of Language-Conditioned Policies for Robot Manipulation”. *arXiv preprint arXiv:2212.04573*, 2022.
268. Y. Zhou, Y. Aytar, and K. Bousmalis. “Manipulator-Independent Representations for Visual Imitation”. *arXiv preprint arXiv:2103.09016*, 2021.
269. Y. Zhu, J. Wong, A. Mandlekar, and R. Martín-Martín. “robosuite: A Modular Simulation Framework and Benchmark for Robot Learning”. In: *arXiv preprint arXiv:2009.12293*. 2020.