CLOSING THE PERCEPTION-ACTION LOOP: TOWARDS GENERAL-PURPOSE ROBOT AUTONOMY

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Yuke Zhu
August 2019

This dissertation is online at: http://purl.stanford.edu/jg446vg2066

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Fei-Fei Li, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Jeannette Bohg

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Emma Brunskill

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Abstract

Robots and autonomous systems have been playing a significant role in the modern economy. Custom-built robots have remarkably improved productivity, operational safety, and product quality. However, these robots are usually programmed for specific tasks in narrow domains, unable to quickly adapt to new tasks and novel situations. The advent of affordable, lightweight, and flexible robot hardware has opened up opportunities for scaling up robot autonomy to an unprecedented level. A major challenge for the new robot hardware to operate in everyday settings is to handle the constant variability and uncertainty of the real world. To tackle this challenge, we have to address the synergy between perception and action: on the one hand, the robot's perception guides its action adaptively, and on the other hand, its action gives rise to new perceptual information for decision making. I argue that a vital step towards a general-purpose robot autonomy is to integrate perception and action in a tight loop.

Emerging computational tools in artificial intelligence have demonstrated promising successes and constitute ideal candidates to enhance robots' perception and control in unstructured environments. The embodied nature of robotics compels us to move beyond the existing paradigm of learning from disembodied datasets and inspires us to develop novel algorithms that take into account the physical hardware and the complex system dynamics.

This dissertation demonstrates our research that builds methods and mechanisms for generalizable robot perception and control. Our work illustrates that the tight coupling of perception and action facilitates robots to interact with the unstructured world through their senses, to flexibly perform a wide range of tasks, and to adaptively learn new tasks. Our findings show that dissecting the perception-action loop at three levels of abstraction, from the low-level motor skills to high-level task understanding, effectively prompts the robustness and generalization of robot behaviors. Laying out our research work that attends to tasks with growing complexity unfolds our roadmap towards the holy-grail goal: building long-term, general-purpose robot autonomy in the real world.

Acknowledgments

First and foremost, I am deeply indebted to my doctoral advisor, Fei-Fei Li, who has taught me to be a critical thinker, a creative researcher, and most importantly a good person. Fei-Fei guides me to focus on big problems, to carry on from the setbacks, and to persistently challenge myself to achieve greater things beyond my imagination. From Fei-Fei, I have learned the importance of passion, grit, and dedication in research and in life. Her mentoring has made me realize the long-lasting positive influence a professor could have on his/her students, which is one of the main reasons that lead me to pursue a career in academia. Fei-Fei will continue to be my role model as an exceptional educator and an impactful researcher. I could not have dreamed about having a better Ph.D. advisor.

It has also been a tremendous honor to have the opportunities to work with other faculty at Stanford, including my doctoral co-advisor, Silvio Savarese, my thesis committee members Jeannette Bohg, Emma Brunskill, and Dan Yamins, my research collaborators Michael Bernstein, Chris Ré, Juan Carlos Niebles, and Marco Pavone. They have demonstrated what serious scientists and deep thinkers are like, and will continue to have a profound influence on my research philosophy.

Sincere gratitude should be addressed to Greg Mori and Oliver Schulte for bringing me into the thrill of AI research and guiding me writing my first academic publications in my undergraduate years. I also owe enormous thanks to Nando de Freitas for being a wonderful mentor and for supporting my academic job search.

I feel immensely fortunate to have worked with a group of brilliant minds, particularly, members of the Stanford Vision and Learning Lab: Alexandre Alahi, Chris Baldassano, Lamberto Ballan, Shyamal Buch, Chris Choy, Jia Deng, Alireza Fathi, Timnit Gebru, Michelle Greene, Judy Hoffman, Andrej Karpathy, Ranjay Krishna, Tian Lan, Li-Jia Li, Joseph Lim, Olga Russakovsky, Guido Pusiol, Vignesh Ramanathan, Kevin Tang, Jonathan Krause, Yu Xiang, Serena Yeung, and Amir Zamir. I cherish the countless intellectual discussions, the sleepless nights before deadlines, the fun activities after deadlines, and so many more memorable moments.

I would like to extend special thanks to the fellow students who started the People, AI & Robots

group with me in the third year of my Ph.D.: Linxi Fan, Kuan Fang, Animesh Garg, De-An Huang, Andrey Kurenkov, Michelle Lee, Ajay Mandlekar, Roberto Martín-Martín, Suraj Nair, William Shen, and Danfei Xu. It has always been inspirational and fulfilling to work with them to explore the most challenging open questions at the forefront of robot perception and learning together.

Most importantly, I could not be more grateful to the unconditional care, love, and support from my family. My parents gave me early exposures to computer programming and information technology when I was a kid. I could never thank them enough for always giving me the freedom to look for my keen interests, for their unwavering belief in me, and for being proud of me as their son. I count myself extremely lucky that I found computer science as my genuine passion at an early age, and have turned this passion into a life-time dedication. Finally, I would like to express my deepest gratitude to my wife Xisai, who has been my greatest friend and best companion. It is wonderful to have her with me through every step of my Ph.D. journey. Without her, it could be a lot tougher. She is the source of inspiration, enthusiasm, and happiness of my life!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

From hoes to looms, humans have innovated all sorts of tools and machines to reduce physical labor and manual work. Since the dawn of the Industrial Revolution, scientists and engineers have begun to develop in earnest autonomous machines, i.e., *robots*, that can carry out tasks with minimum human supervision. To date, custom-built robots have remarkably improved productivity, operational safety, and product quality. However, their applications are limited to narrow domains. Building general-purpose robots that are capable of performing day-to-day tasks in the real world will bring about a new form of automation that augments humans' physical and cognitive abilities.

In recent years, tremendous progress has been made in building affordable, lightweight, and flexible robot hardware. The advent of new hardware platforms has opened up opportunities for scaling up robot autonomy to an unprecedented level. To empower the new robot hardware to operate in everyday settings, people have resorted to artificial intelligence for principles and methods. A major challenge for robots to become commonplace in our homes, schools, and offices, is to deal with the constant variability and uncertainty of unstructured environments. The vast diversity of objects and scenes, the lack of perfect knowledge about the surroundings, and unpredictable dynamics all aggravate the difficulty of robotic perception and control in practice. To tackle this challenge, it is essential to address the synergic relationship between perception and action that is ubiquitously seen in real-world robotic systems (see Figure 1.1). On one hand, the robot's perception guides its action adaptively, and on the other hand, its action gives rise to new perceptual information for decision making. Therefore, I argue that a vital step towards general-purpose robot autonomy is to close the perception-action loop, where sensing and control are coherently and seamlessly integrated.

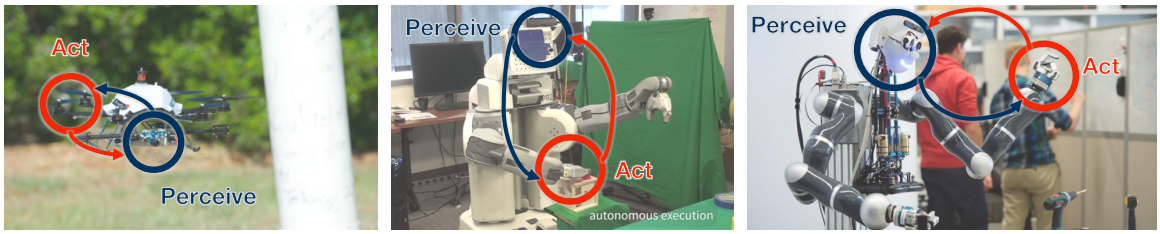


Figure 1.1: The ubiquitous presence of the perception-action loop in real-world robotic systems. Image credits from left to right: [207], [140], and [21].

Recent advances in artificial intelligence have created a set of new tools for perception and control. We have witnessed tremendous progress in computer vision and machine learning in the past few years. Data-driven models, such as deep neural networks [135], fueled by large-scale datasets [203] have elevated standardized visual recognition performances, such as image classification [91, 222, 128] and object detection [77, 90, 196], to the human level. Deep reinforcement learning methods have mastered ATARI video games [161] and defeated the human world champion of Go [219]. These emerging computational tools demonstrate promising successes and constitute ideal candidates to enhance robots' perception and control in unstructured environments. However, robotics, as an impactful arena for machine learning and computer vision, presents a set of unique and pressing technical challenges. The interactive nature of robotics compels us to move beyond the existing paradigm of learning from disembodied datasets and to develop novel algorithms that take into account the embodiment of the physical hardware and the complex system dynamics.

This dissertation demonstrates our research that builds methods and mechanisms for generalizable robot perception and control. We leverage machine learning and computer vision techniques to develop new algorithms for robotic manipulation and navigation. Our work illustrates that the tight coupling of perception and action facilitates robots to interact with the unstructured world through their senses, to flexibly perform a wide range of tasks, and to adaptively learn new tasks. Our findings show that dissecting the perception-action loop at different levels of abstraction, from the low-level motor skills to high-level task understanding, effectively prompts the robustness and generalization of robot behaviors. Specifically, this dissertation examines three complementary questions at increasing levels of abstraction:

- How can robots learn motor skills from raw sensory data?
- How can robots reuse knowledge from past experience to effectively learn new tasks?

Table 1.1: High-Level Summary of Dissertation Structure

	Chapters	Perception Modality	Action Abstraction	Learning Algorithm
Part I: Primitive Skills	2, 3	real-world sensory data	joint torque	reinforcement, imitation, self-supervised learning
Part II: Sequential Tasks	4, 5	interactive visual environment	high-level command	transfer learning
Part III: Hierarchical Tasks	6, 7	unstructured video data	task structure	meta-learning

- How can visual data supply knowledge for robots to perform complex tasks?

To address these questions, our work draws on theories and methods from machine learning, computer vision, and robotics, along with inspiration from cognitive science and psychology. Laying out our research work that attends to tasks with growing complexity unfolds our roadmap towards the holy-grail goal: building long-term, general-purpose robot autonomy in the real world.

1.2 Thesis Outline

This dissertation is structured into three main parts, in response to the three questions above. Each part consists of two chapters. Part I focuses on learning *primitive skills* from raw sensory data. Chapter 2 and Chapter 3 introduce two types of methods to learn dexterous manipulation behaviors from raw sensory information. Once we have learned a diverse set of primitive skills, Part II investigates how to sequentially compose them to perform *sequential tasks* in interactive environments. To evaluate sequential tasks in a richer and more realistic setting, Chapter 4 introduces a new simulated platform that we designed for visual learning of embodied agents. With this platform as testbed, Chapter 5 discusses a transfer learning algorithm that shares knowledge across tasks for faster learning of new tasks. Finally, Part III moves on to complex *hierarchical tasks*, where simple

sequential composition is inefficacious. We propose to exploit the task compositionality to reduce the problem size and to extract such task structures from video data. Chapter 6 and Chapter 7 cover two meta-learning models that learn the latent compositional structures of these complex tasks from video demonstrations. The ordering of these parts reflects a progression of abstraction and a growing task complexity. Table 1.1 offers a high-level summary of the dissertation structure, including the perception modality, action abstraction, and core machine learning techniques used in each part of this dissertation. We expand our discussion of each part in greater detail as follows.

1.2.1 Learning primitive skills from raw sensory data

Primitive motor skills [101, 180, 183], such as grasping, reaching, pushing, constitute the fundamental building blocks for more sophisticated robot behaviors. The unstructured world demands a robot to develop these primitive skills to act upon its senses. A central challenge here is to build good representations of the world from raw sensory data, based on which basic motor skills can be learned. Our research investigates how to use deep neural networks to learn robust representations from high-dimensional perceptual inputs. In computer vision, it has been shown that deep neural networks can learn good representations for visual recognition from disembodied datasets [203]. However, the physical embodiment of robots requires the representations to be informative about their interactions in the environments. Our work demonstrates that robust representations of raw sensory data can be learned through a robot's own experience in the physical world [57, 136, 272].

In this part, we highlight two lines of work that leverage different learning mechanisms to build robust representations for robot control. Chapter 2 describes our work in combining reinforcement learning and imitation learning for end-to-end visuomotor learning [272] (see Figure 1.2). The algorithm, built upon the ideas of adversarial learning [95] and curriculum learning, can learn complex skills from raw pixels with the aid of a handful of human demonstrations. We show that it can effectively learn action-informed representations for sensorimotor skills, and the visuomotor policies learned in physical simulation can be successfully deployed to real hardware. Inspired by this result, we develop RoboTurk [154, 155], a cloud-based crowdsourcing

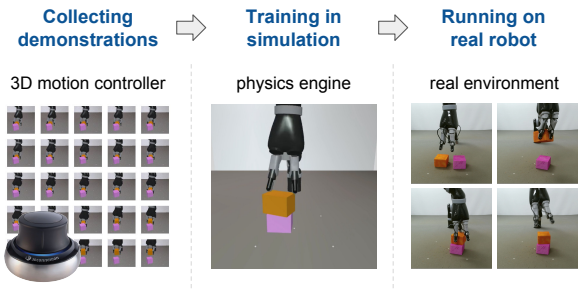


Figure 1.2: Learning motor skills with reinforcement and imitation learning [272]

platform to collect teleoperated demonstrations from remote workers, and Surreal [56], an open-source framework for distributed deep reinforcement learning.

Chapter 2 centers around visual perception as the primary sensory modality. An intelligent robot should eventually interact with the environments through a combination of multimodal senses, including vision, touch, sound, etc. Rather than end-to-end learning from perception to control, Chapter 3 explores an alternative learning paradigm to build multimodal representations. We introduce a self-supervised algorithm that enables a robot to learn a compact representation of haptic and visual feedback without manual labelling [136]. We demonstrate the effectiveness of these representations in contact-rich manipulation tasks such as high-precision assembly.

1.2.2 Reusing past experience for fast learning

Primitive motor skills, learned by the former methods, provide a large repertoire of building blocks for composing sophisticated robot behaviors. Practical tasks often include a collection of primitive skills. For example, a task of making salad would involve a sequence of lower-level skills, such as cutting vegetables, washing bowls, placing dressings, tossing ingredients, etc. We thus study how to perform these sequential tasks by composing the primitive skills one-by-one.

A significant challenge stems from the large space of possible tasks, making it intractable to learn every instance of new tasks from scratch. We draw upon intuition of human learning, where prior experience from similar tasks leads to faster adaptation of new tasks [53, 133, 185]. It inspires us to develop transfer learning algorithms that share knowledge from tasks to similar ones.

Transfer learning has been widely studied in various machine learning contexts [176]. It has been a common practice in visual recognition to warm-start the training of new models with pre-trained parameters [100, 265]. In contrast, our work focuses on transfer learning between goal-directed policies of sequential tasks. Prior work on policy transfer has focused on simplistic domains and limited pool of tasks [13, 178, 204, 235]. To study this problem in a broader and more realistic setup, we developed AI2-THOR, a 3D simulated platform that integrates a physical engine with a photorealistic graphics renderer (see Figure 1.3). This platform enables a virtual robot to move and to interact with objects in hundreds of indoor scenes,

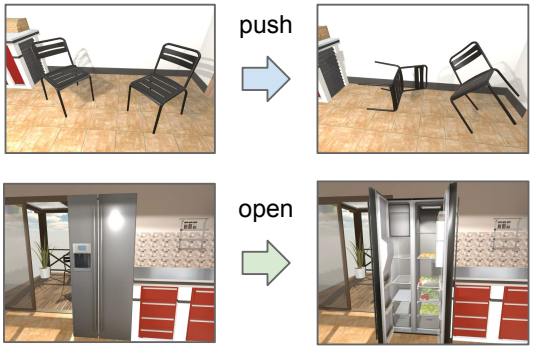


Figure 1.3: Example interactions in simulated environments of AI2-THOR [271]

making it an ideal testbed with a large number of tasks.

Chapter 4 starts with an overview of the AI2-THOR platform [122]. We also talk about our work on target-driven visual navigation [271] using this platform. Our key insight is to achieve generalization and fast adaptation through a special parameterization of the navigation policies. It allows the majority of model parameters to be shared across different navigation targets, different scenes, and from simulation to reality.

Once the robots learn how to navigate and manipulate, we draw our attention to the high-level task planning problem while abstracting away the underlying motor control. We examine the task of visual semantic planning [268] in Chapter 5. Our goal is to learn goal-directed policies that produce a sequence of high-level commands given a task (e.g., search for the key and put it on the tabletop), where the task goal is specified by a reward function. The core technique is a transfer learning model that decouples the environment dynamics from the task embeddings via a linear factorization of the reward function. This decoupling makes it possible to quickly synthesize policies for new tasks from trained policies of related tasks.

1.2.3 Learning complex tasks from visual data

Finally, we investigate high-level, long-horizon tasks that require a prolonged interaction with the environment, e.g., "preparing dinner" or "driving to San Francisco". Our key observation is that such tasks usually follow a latent compositional structure, where the entire task can be hierarchically decomposed into simpler and shorter subtasks for divide-and-conquer. For these hierarchical tasks, the sequential composition of primitive skills is no longer tractable due to the exponential growth of possible combinations. Therefore, we need to level up the abstraction and reduce the problem size.

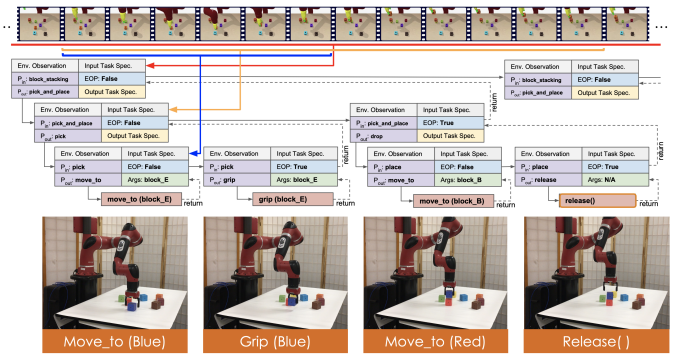


Figure 1.4: Modeling complex tasks with compositional structures [260]

Part III of this dissertation is dedicated to our work in compositional task learning. Visual data, such as instruction videos, provide a vast amount of accessible yet untapped resources for learning complex real-world tasks. We inquire into how to extract the compositional task structures from video demonstrations. We cast it as a meta-learning problem, where the goal is to perform one-shot

visual imitation, i.e., learning to perform a novel task from a single video demonstration of the task. Prior work on one-shot imitation learning has modeled demonstrations as a flat sequence [52]. In contrast, we develop algorithms that capture the compositional structures of complex tasks.

Chapter 6 introduces Neural Task Programming (NTP) [260], where we use neural program induction to parse a visual demonstration into an intermediate task representation in the form of latent hierarchical programs (see Figure 1.4). From these compositional task representations, a closed-loop policy can be constructed for robots to perform the demonstrated task. NTP has shown promising results in learning novel tasks. In comparison to baselines without the compositional modeling, NTP achieves substantially better generalization performances with a smaller amount of training data. Nonetheless, NTP has used generic neural network modules, i.e., black-box models where the inner workings are opaque. It compels us to consider the follow-up question: can we use more explicit task representations as the inductive bias to better capture the task structures?

Chapter 7 introduces Neural Task Graphs (NTG) [99], where we propose to use conjugate task graphs as the intermediate task representations inside the neural network modules. We show that by explicitly incorporating compositionality as the inductive bias inside the end-to-end deep networks, NTG achieves stronger generalization than black-box NTP models.

1.3 Other Doctoral Research Work

This dissertation is primarily based on my doctoral research published in the last three years of my Ph.D. study. A substantial amount of the earlier years, has dedicated to a variety of visual perception problems towards a deeper understanding of the unstructured world. These works can be roughly categorized into three parallel themes: 1) structured scene and activity understanding [126, 134, 249, 261], 2) joint reasoning about vision and language modalities [126, 269, 270], and 3) knowledge-based visual reasoning [267, 273].

My experience working on a variety of robotics problems has convinced me that perception is fundamentally a weak link in the perception-action loop — the lion's share of complexity in many practical problems in robotics has deeply rooted from the inadequate perception capabilities. Thus, my earlier works in visual perception have honed my research intuitions and paved the ground for me to work on the perception-action loop in the later stage of my Ph.D.

Chapter 2

Reinforcement and Imitation Learning

2.1 Introduction

Recent advances in deep reinforcement learning (RL) have performed very well in several challenging domains such as video games [161] and Go [219]. For robotics, RL in combination with powerful function approximators such as neural networks provides a general framework for designing sophisticated controllers that would be hard to handcraft otherwise. Reinforcement learning methods have a long history in robotics control but have typically been used with low-dimensional movement representations [45, 119]. The last few years have seen a growing number of successful demonstrations of deep RL for robotic manipulation using model-based (e.g. [140, 142, 262]) and model-free techniques (e.g. [32, 80, 190]), both in simulation and on real hardware. Nevertheless, end-to-end learning of visuomotor controllers for long-horizon and multi-stage manipulation tasks using model-free RL techniques remains a challenging problem.

Developing RL agents for robotics requires overcoming several significant challenges. Policies for robotics must transform multi-modal and partial observations from noisy sensors, such as cameras, into coordinated activity of many degrees of freedom. At the same time, realistic tasks often come with contact-rich dynamics and vary along multiple dimensions (visual appearance, position, shapes, etc.), posing significant generalization challenges. Model-based methods can have difficulties handling such complex dynamics and large variations. Directly training model-free methods on real robotics hardware can be daunting due to the high sample complexity. The difficulty of real-world RL training is compounded by safety considerations as well as the difficulty of accessing information about the state of the environment (e.g., the position of an object) to define a reward function. Finally, even in simulation when perfect state information and large amounts of training

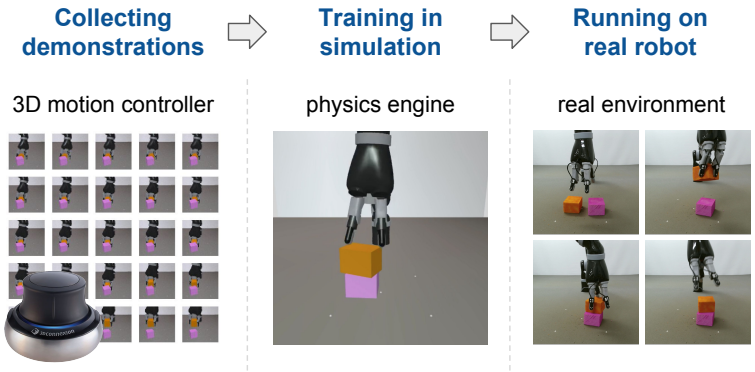


Figure 2.1: Our proposal of a principled robot learning pipeline. We used 3D motion controllers to collect human demonstrations of a task. Our reinforcement and imitation learning model leveraged these demonstrations to facilitate learning in a simulated physical engine. We then performed sim2real transfer to deploy the learned visuomotor policy to a real robot.

data are available, exploration can be a significant challenge, especially for on-policy methods. This is partly due to the often high-dimensional and continuous action space, but also due to the difficulty of designing suitable reward functions.

In this chapter, we present a model-free deep RL method that can solve a variety of robotic manipulation tasks directly from pixel input. Our key insights are 1) to reduce the difficulty of exploration in continuous domains by leveraging a handful of human demonstrations; 2) to leverage several new techniques that exploit privileged and task-specific information during training only which can accelerate and stabilize the learning of visuomotor policies in multi-stage tasks; and 3) to improve generalization by increasing the diversity of the training conditions. As a result, the policies work well under significant variations of system dynamics, object appearances, task lengths, etc. Furthermore, we demonstrate promising preliminary results for two tasks, where the policies trained in simulation achieve zero-shot transfer to a real robot.

We evaluate our method on six manipulation tasks, including stacking, pouring, etc. The set of tasks includes multi-stage and long-horizon tasks, and they require full 9-DoF joint velocity control directly from pixels. The controllers need to be able to handle significant shape and appearance variations.

To address these challenges, our method combines imitation learning with reinforcement learning into a unified training framework. Our approach utilizes demonstration data in two ways: first, it uses a hybrid reward that combines the task reward with an imitation reward based on Generative Adversarial Imitation Learning [95]. This aids with exploration while still allowing the final

controller to outperform the human demonstrator on the task. Second, it uses demonstration trajectories to construct a curriculum of states along which to initialize the episodes during training. This enables the agent to learn about later stages of the task earlier in training, facilitating the solving of long tasks. As a result, our approach solves all six tasks, which neither the reinforcement learning nor imitation learning baselines can solve alone.

To sidestep the constraints of training on real hardware we embrace the sim2real paradigm which has recently shown promising results [103, 204, 236]. Through the use of a physics engine and high-throughput RL algorithms, we can simulate parallel copies of a robot arm to perform millions of complex physical interactions in a contact-rich environment while eliminating the practical concerns of robot safety and system reset. Furthermore, we can, during training, exploit privileged and task-specific information about the true system state with several new techniques, including learning policy and value in separate modalities, an object-centric GAIL discriminator, and auxiliary tasks for visual modules. These techniques stabilize and speed up policy learning, without imposing any constraints on the system at test time.

Finally, we diversify training conditions such as visual appearance, object geometry, and system dynamics. This improves both generalization with respect to different task conditions as well as transfer from simulation to reality.

We use the same model and the same algorithm with only small task-specific modifications of the training setup to learn visuomotor controllers for six diverse robot arm manipulation tasks. As illustrated in Figure 2.1 this instantiates a visuomotor learning pipeline going from collecting human demonstration to learning in simulation, and back to real-world deployment via sim2real policy transfer.

2.2 Related work

Reinforcement learning methods have been extensively used with low-dimensional policy representations such as movement primitives to solve a variety of control problems both in simulation and in reality. Three classes of RL algorithms are currently dominant for continuous control problems: guided policy search methods (GPS; [141]), value-based methods such as the deterministic policy gradient (DPG; [94, 146, 220]) or the normalized advantage function (NAF; [81]) algorithm, and trust-region based policy gradient algorithms such as trust region policy optimization (TRPO) and proximal policy optimization (PPO). TRPO [213] and PPO [214] hold appeal due to their robustness to hyperparameter settings as well as their scalability [92] but the lack of sample efficiency makes

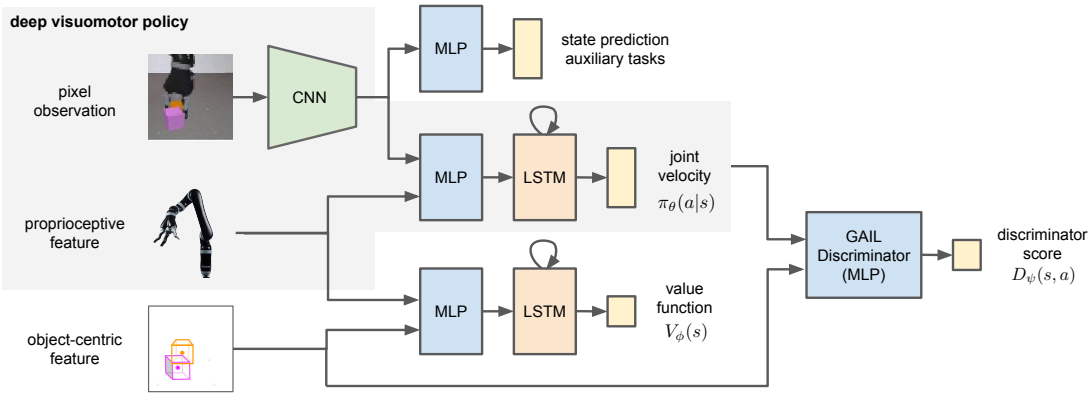


Figure 2.2: Model overview. The core of our model is the deep visuomotor policy, which takes the camera observation and the proprioceptive feature as input and produces the next joint velocities.

them unsuitable for training directly on robotics hardware.

GPS [141] has been used e.g. by [140], [262] and [32] to learn visuomotor policies directly on a real robotics hardware after a network pretraining phase. [84] and [131] use GPS for learning controllers for robotic hand models. Value-based methods have been employed, e.g. by [80] who use NAF to learn a door opening task directly on a robot while [190] demonstrate how to solve a stacking problem efficiently using a distributed variant of DPG.

The idea of using large-scale data collection for training visuomotor controllers has been the focus of [142] and [188] who train a convolutional network to predict grasp success for diverse sets of objects using a large dataset with 10s or 100s of thousands of grasp attempts collected from multiple robots in a self-supervised setting.

An alternative strategy for dealing with the data demand is to train in simulation and transfer the learned controller to real hardware, or to augment real-world training with synthetic data. [205] learn simple visuomotor policies for a Jaco robot arm and transfer to reality using progressive networks [204]. [246] minimize the reality gap by relying on depth. [236] use visual variations to learn robust object detectors that can transfer to reality; [103] combine randomization with supervised learning. [26] augment the training with simulated data to learn grasp prediction of diverse shapes.

Suitable cost functions and exploration strategies for control problems are challenging to design, so demonstrations have long played an important role. Demonstrations can be used to initialize policies, design cost functions, guide exploration, augment the training data, or a combination of these. Cost functions can be derived from demonstrations either via tracking objectives (e.g. [84]) or via inverse RL (e.g. [25, 64]), or, as in our case, via adversarial learning [95]. When expert actions

or expert policies are available, behavioral cloning can be used ([52, 103, 192]). Alternatively, expert trajectories can be used as additional training data for off-policy algorithms such as DPG (e.g. [244]). Most of these methods require observation and/or action spaces to be aligned between the robot and demonstrations. Recently, methods for third person imitation have been proposed (e.g. [65, 150, 217]).

Concurrently with our work several works have presented results on manipulation tasks. [164, 193] both use human demonstrations to aid exploration. [164] extends the DDPGfD algorithm [244] to learn a block stacking task on a position-controlled arm in simulation. [193] use the demonstrations with a form of behavioral cloning and data augmentation to learn several complex manipulation tasks. In both cases, controllers observe a low-level state feature and these methods inherently require aligned state and action spaces with the demonstrations. In contrast, our method learns end-to-end visuomotor policies without reliance on demonstrator actions. Thus, it can utilize demonstrations when raw demonstrator actions are unknown or generated by a different body. [186] and [181] address the transfer from simulation to reality, focusing on randomizing visual appearance and robot dynamics respectively. [181] transfer a block-pushing policy operating from state features to a 7-DoF position controlled Fetch robotics arm. [186] consider different tasks using visual input with end-effector position control. These concurrent works have each introduced a subset of techniques that our model employs. This work, developed independently from concurrent works, integrates several new techniques into one coherent method. Our experimental results demonstrate that good performances come from the synergy of these combined techniques.

2.3 Model

Our goal is to learn a visuomotor policy with deep neural networks for robot manipulation tasks. The policy takes both an RGB camera observation and a proprioceptive feature vector that describes the joint positions and angular velocities. These two sensory modalities are also available on the real robot, allowing us to train in simulation and subsequently transfer the learned policy to the robot without modifications. Figure 5.3 provides an overview of our model. The deep visuomotor policy encodes the pixel observation with a convolutional network (CNN) and the proprioceptive feature with a multilayer perceptron (MLP). The features from these two modules are concatenated and passed to a recurrent long short term memory (LSTM) layer before producing the joint velocities (control commands). The whole network is trained end-to-end. We start with a brief review of the basics of generative adversarial imitation learning (GAIL) and proximal policy optimization (PPO).

Our model extends upon these two methods for visuomotor skills.

2.3.1 Background: GAIL and PPO

Imitation learning (IL) is the problem of learning a behavior policy by mimicking a set of demonstrations. Here we assume that human demonstrations are provided as a dataset of state-action pairs $\mathscr{D} = \{(s_i, a_i)\}_{i=1...N}$. Some IL methods cast the problem as one of supervised learning, i.e., behavior cloning. These methods use maximum likelihood to train a parameterized policy $\pi_\theta : \mathscr{S} \to \mathscr{A}$, where $\mathscr{S}$ is the state space and $\mathscr{A}$ is the action space, such that $\theta^* = \arg\max_\theta \sum_N \log \pi_\theta(a_i|s_i)$. The behavior cloning approach works effectively when demonstrations are abundant [201]. However, as robot demonstrations can be costly and time-consuming to collect, we aim for a method that can learn from a handful of demonstrations. GAIL [95] uses demonstration data efficiently by allowing the agent to interact with the environment and learn from its own experiences. Similar to Generative Adversarial Networks (GANs) [79], GAIL employs two networks, a policy network $\pi_\theta : \mathscr{S} \to \mathscr{A}$ and a discriminator network $D_\psi : \mathscr{S} \times \mathscr{A} \to [0,1]$. It uses a min-max objective function similar to that of GANs:

$$\min_\theta \max_\psi \mathbb{E}_{\pi_E}[\log D_\psi(s,a)] + \mathbb{E}_{\pi_\theta}[\log(1 - D_\psi(s,a))], \tag{2.1}$$

where π_E denotes the expert policy that generated the demonstration trajectories. This objective encourages the policy π_θ to have an occupancy measure close to that of the expert policy.

In this work we train π_θ with policy gradient methods to maximize the discounted sum of the reward function $r_{gail}(s_t, a_t) = -\log(1 - D_\psi(s_t, a_t))$, clipped at a max value of 10. In continuous domains, trust region methods greatly stabilize policy training. GAIL was originally presented in combination with TRPO [213] for updating the policy. Recently, PPO [214] has been proposed as a simple and scalable approximation to TRPO. PPO only relies on first-order gradients and can be easily implemented with recurrent networks in a distributed setting [92]. PPO implements an approximate trust region that limits the change in the policy per iteration. This is achieved via a regularization term based on the Kullback-Leibler (KL) divergence, the strength of which is adjusted dynamically depending on actual change in the policy in past iterations.

2.3.2 Reinforcement and Imitation Learning Model

Hybrid IL/RL Reward

Shaping rewards are a popular means of facilitating exploration. Although reward shaping can be very effective it can also lead to suboptimal solutions [166]. Hence, we design the task rewards as sparse piecewise constant functions based on the different stages of the respective tasks. For example, we define three stages for the *block stacking* task, including *reaching*, *lifting*, and *stacking*. Reward change only occurs when the task transits from one stage to another. In practice, we find defining such a sparse multi-stage reward easier than handcrafting a dense shaping reward and less prone to producing suboptimal behaviors. Training agents in continuous domains with sparse or piecewise constant rewards is challenging. Inspired by reward augmentation as described in [144] and [158], we provide additional guidance via a hybrid reward function that combines the imitation reward r_{gail} with the task reward r_{task}:

$$r(s_t, a_t) = \lambda r_{gail}(s_t, a_t) + (1-\lambda) r_{task}(s_t, a_t)\ \lambda \in [0,1]. \tag{2.2}$$

Maximizing this hybrid reward can be interpreted as simultaneous reinforcement and imitation learning, where the imitation reward encourages the policy to generate trajectories closer to demonstration trajectories, and the task reward encourages the policy to achieve high returns on the task. Setting λ to either 0 or 1 reduces this method to the standard RL or GAIL setups. In our experiments, with a balanced contribution of these two rewards the agents can solve tasks that neither GAIL nor RL can solve alone. Further, the final agents achieve higher returns than the human demonstrations owing to the exposure to task rewards.

Leveraging Physical States in Simulation

The physics simulator we employ for training exposes the full state of the system. Even though such privileged information is unavailable on a real system, we can take advantage of it when training the policy in simulation. We propose four techniques for leveraging the physical states in simulation to stabilize and accelerate learning (1) the use of a curriculum derived from demonstration states, (2) the use of privileged information for the value function (baseline), (3) the use of object-centric features in the discriminator, and (4) auxiliary tasks. We elaborate these four techniques as follows:

1. Demonstration as a curriculum. The problem of exploration in continuous domains is exacerbated by the long duration of realistic tasks. Previous work indicates that shaping the distribution of

start states towards states where the optimal policy tends to visit can greatly improve policy learning [109, 190]. We alter the start state distribution with demonstration states. We build a curriculum that contains clusters of states in different stages of a task. For instance, we define three clusters for the pouring task, including *reaching the mug*, *grasping the mug*, and *pouring*. During training, with probability ε, we then start an episode from a random initial state, and with probability $1-\varepsilon$ we uniformly select a cluster and initialize the episode with a demonstration state from that cluster. This is possible since our simulated system is fully characterized by the physical states.

2. Learning value functions from states. PPO uses a learnable value function V_ϕ to estimate the advantage required to compute the policy gradient. During training, each PPO worker executes the policy for K steps and uses the discounted sum of rewards and the value as an advantage function estimator $\hat{A}_t = \sum_{i=1}^{K} \gamma^{i-1} r_{t+i} + \gamma^{K-1} V_\phi(s_{t+K}) - V_\phi(s_t)$, where γ is the discount factor. As the policy gradient relies on the value function to reduce variance, it is beneficial to accelerate learning of the value function. Rather than using pixels as inputs similar to the policy network, we take advantage of the low-level physical states (e.g., the position and velocity of the 3D objects and the robot arm) to train the value V_ϕ with a smaller multilayer perceptron. We find that training the policy and value in two different modalities stabilizes training and reduces oscillation of the agent's performance. This technique has also been been proposed concurrently by [186].

3. Object-centric discriminator. As for the value function, we exploit the availability of the physical states for the GAIL discriminator and provide task specific features as input. We find that object-centric representations (e.g., absolute and relative positions of the objects) provide the salient and relevant signals to the discriminator. The states of the robot arm in contrast lead the discriminator to focus on irrelevant aspects of the behavior of the controller and are detrimental for training of the policy. Inspired by information hiding strategies used in locomotion domains [93, 158], our discriminator only takes the object-centric features as input while masking out arm-related information. The construction of the object-centric representation requires a certain amount of domain knowledge of the tasks. We find that the relative positions of objects and displacements from the gripper to the objects usually provide the most informative characterization of a task. Empirically, we find that our model is not very sensitive to the particular choices of object-centric features, as long as they carry sufficient task-specific information. We provide detailed descriptions in Appendix A.3.

4. State prediction auxiliary tasks. Auxiliary tasks have been shown to be effective in improving the learning efficiency and the final performance of deep RL methods [102]. To facilitate learning visuomotor policies we add a state prediction layer on the top of the CNN module to predict the

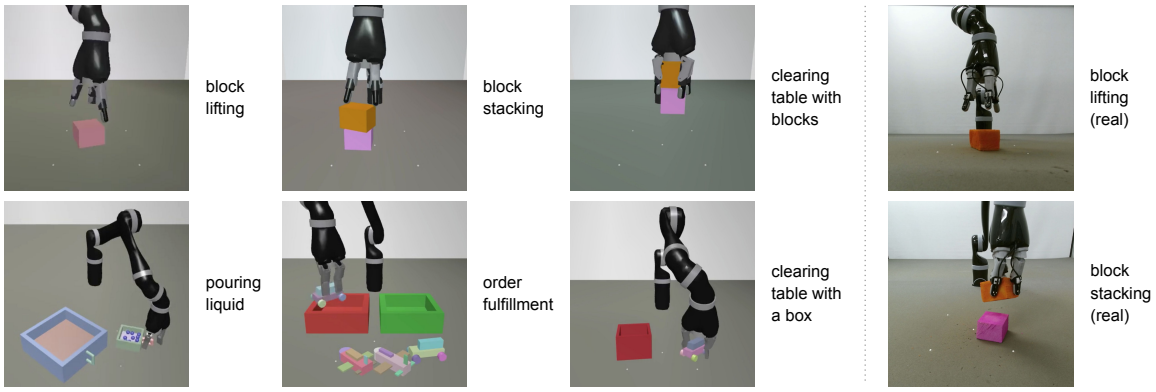
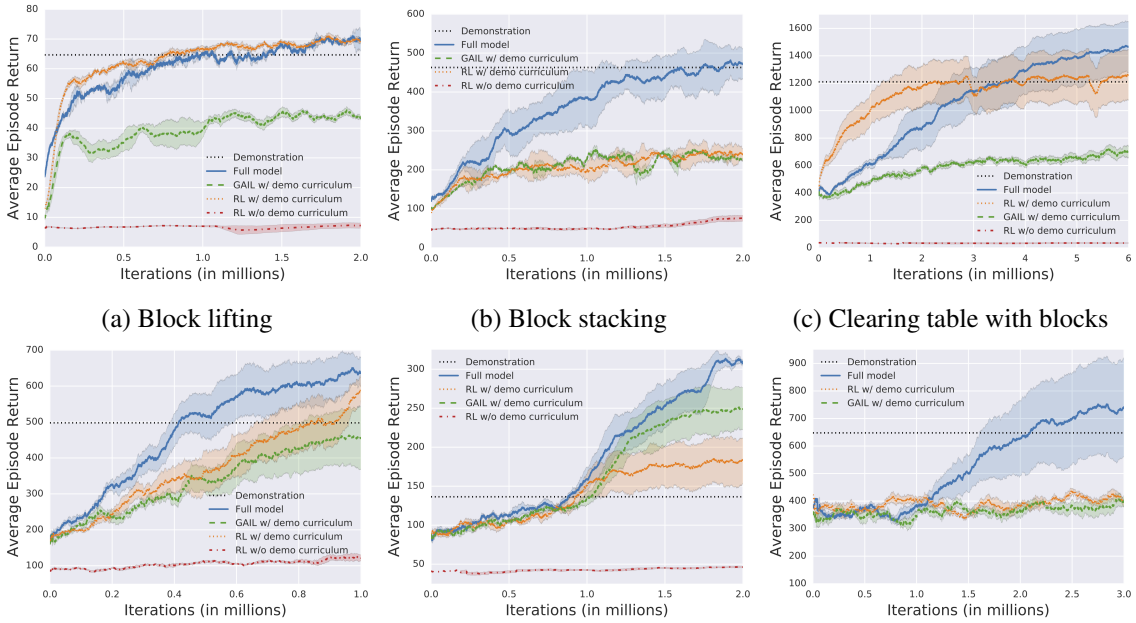


Figure 2.3: Visualizations of the six manipulation tasks in our experiments. The left column shows RGB images of all six tasks in the simulated environments. These images correspond to the actual pixel observations as input to the visuomotor policies. The right column shows the two tasks with color blocks on the real robot.

locations of objects from the camera observation. We use a fully-connected layer to regress the 3D coordinates of objects in the task, minimizing the ℓ_2 loss between the predicted and ground-truth object locations. The auxiliary tasks are not required for our model to learn good visuomotor policies; however, adding the additional supervision can often accelerate the training of the CNN module.

Sim2Real Policy Transfer

We perform policy transfer experiments on a real-world Kinova Jaco robot arm. The simulation was manually adjusted to roughly match the appearance and dynamics of the laboratory setup: a Kinect camera was visually calibrated to match the position and orientation of the simulated camera, and the simulation's dynamics parameters were manually adjusted to match the dynamics of the real arm. Instead of using professional calibration equipment, our approach to sim2real policy transfer relies on domain randomization of camera position and orientation [103, 236]. In contrast to some previous works our trained policies do not rely on any object position information or intermediate goals but rather learn a mapping end-to-end from raw pixel input joint velocities. In addition, to improve the robustness of our controllers to latency effects on the real robot, we also fine-tune our policies while subjecting them to action dropping. A detailed description is available in Appendix A.2.



(a) Block lifting (b) Block stacking (c) Clearing table with blocks

(d) Clearing table with a box (e) Pouring liquid (f) Order fulfillment

Figure 2.4: Learning efficiency of our reinforcement and imitation model against baselines. The plots are averaged over 5 runs with different random seeds. All the policies use the same network architecture and the same hyperparameters (except λ).

2.4 Experiments

Here we demonstrate that our approach offers a flexible framework to visuomotor policy learning. To this end we evaluate its performance on the six manipulation tasks illustrated in Figure 2.3. We provide additional qualitative results in this video: `https://youtu.be/ED18SQUNjj0`.

2.4.1 Environment Setup

We use a Kinova Jaco arm that has 9 degrees of freedom: six arm joints and three actuated fingers. The robot arm interacts with a diverse set of objects on a tabletop. The visuomotor policy controls the robot by setting the joint velocity commands, producing 9-dimensional continuous velocities in the range of $[-1, 1]$ at 20Hz. The proprioceptive features consist of the positions and angular velocities of the arm joints and the fingers. Visual observations of the table-top scene are provided via a suitably positioned real-time RGB camera. The proprioceptive features and the camera observations are available in both simulation and real environments thus enabling policy transfer. The physical environment is simulated in the MuJoCo physics simulator [237].

We use a large variety of objects, ranging from basic geometric shapes to procedurally generated 3D objects built from ensembles of primitive shapes. We increase the diversity of objects by randomizing various physical properties, including dimension, color, mass, friction, etc. We collect demonstrations using a SpaceNavigator 3D motion controller, which allows us to operate the robot arm with a position controller, and gather 30 episodes of demonstration for each task including observations, actions, and physical states. As each episode takes less than a minute to complete, demonstrating each task can be done within half an hour.

2.4.2 Robot Arm Manipulation Tasks

Figure 2.3 shows the six manipulation tasks in our experiments. The first column shows the six tasks in simulated environments, and the second column shows the real-world setup of the block lifting and stacking tasks. We see obvious visual discrepancies of the same task in simulation and reality. These six tasks exhibit learning challenges to varying degrees. The first three tasks use simple colored blocks, which makes it easy to replicate a similar setup on the real robot. We study sim2real policy transfer with the block lifting and stacking tasks in Section 2.4.4.

Block lifting. The goal is to grasp and lift a randomized block, allowing us to evaluate the model's robustness. We vary several random factors, including the robot arm dynamics (friction and armature), lighting conditions, camera poses, background colors, as well as the properties of the block. Each episode starts with a new configuration with these random factors uniformly drawn from a preset range.

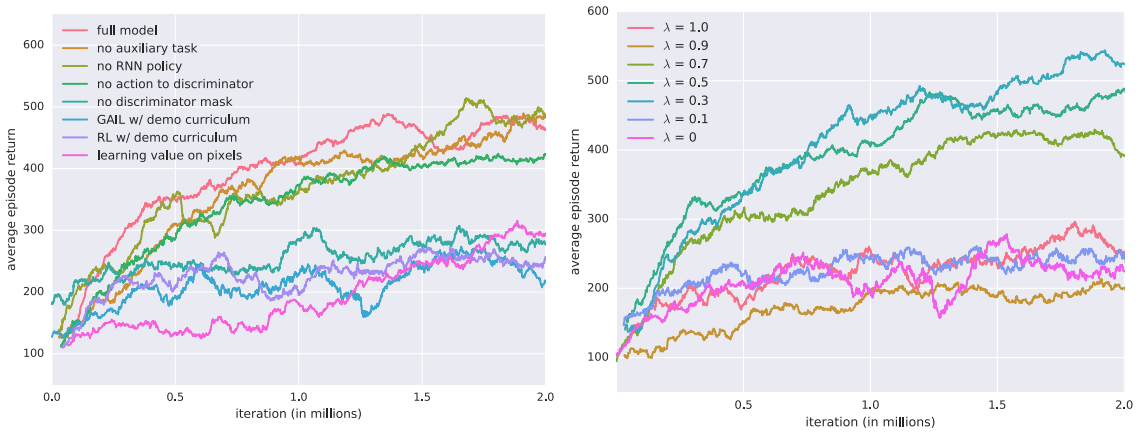
Block stacking. The goal is to stack one block on top of the other block. Together with the block lifting task, this is evaluated in sim2real transfer experiments.

Clearing table with blocks. This task requires lifting two blocks off the tabletop. One solution is to stack the blocks and lift them both together. This task requires longer time and a more dexterous controller, introducing a significant challenge for exploration.

The next three tasks involve a large variety of procedurally generated 3D shapes, making them difficult to recreate in real environments. We use them to examine the model's ability to generalize across object variations in long and complex tasks.

Clearing table with a box. The goal is to clear the tabletop that has a box and a toy car. One strategy is to grasp the toy, put it into the box, and lift the box. Both the box and the toy car are randomly generated for each episode.

Pouring liquid. Modeling and reasoning about deformable objects and fluids is a long-standing challenge in the robotics community [211]. We design a pouring task where we use many small



(a) Ablation study of model components

(b) Model sensitivity to λ values

Figure 2.5: Model analysis in the stacking task. On the left we investigate the impact on performance by removing each individual component from the full model. On the right we investigate the model's sensitivity to the hyperparameter λ that moderates the contribution of reinforcement and imitation.

spheres to simulate liquid. The goal is to pour the "liquid" from one mug to the other container. This task is particularly challenging due to the dexterity required. Even humans struggled to demonstrate the task with our 3D motion controller after extensive practice.

Order fulfillment. In this task we randomly place a variable number of procedurally generated toy planes and cars on the table. The goal is to place all the planes into the green box and all the cars into the red box. This task requires the policy to generalize at an abstract level. It needs to recognize the object categories, perform successful grasps on diverse shapes, and handle tasks with variable lengths.

2.4.3 Quantitative Evaluation

Our full model can solve all six tasks, with only occasional failures, using the same policy network, the same training algorithm, and a fixed set of hyperparameters. On the contrary, neither reinforcement nor imitation alone can solve all tasks. We compare the full model with three baselines which correspond to pure RL, pure GAIL, and RL w/o demonstration curriculum. These baselines use the same setup as the full model, except that we set $\lambda = 0$ for RL and $\lambda = 1$ for GAIL, while our model uses a balanced contribution of the hybrid reward, where $\lambda = 0.5$. In the third baseline, all training episodes start from random initial states rather than resetting to demonstration states. This is a standard RL setup.

We report the mean episode returns as a function of the number of training iterations in Figure 2.4. Our full model achieves the highest returns in all six tasks. The only case where the baseline model is on par with the full model is the block lifting task, in which both the RL baseline and the full model achieved similar levels of performance. We hypothesize that this is due to the short length of the lifting task, where random exploration can provide a sufficient learning signal without the aid of demonstrations. In the other five tasks, the full model outperforms both the reinforcement learning and imitation learning baselines by a large margin, demonstrating the effectiveness of combining reinforcement and imitation for learning complex tasks. Comparing the two variants of RL with and without using demonstration as a curriculum, we see a pronounced effect of altering the start state distribution. We see that RL from scratch leads to very slow learning progress; while initiating episodes along demonstration trajectories enables the agent to train on states from different stages of a task. As a result, it greatly reduces the burden of exploration and improves the learning efficiency. We also report the mean episode returns of human demonstrations in these figures. Demonstrations with the 3D motion controller are imperfect, especially for pouring, and the trained agents exceed the performance of the human operator.

Two findings are noteworthy. First, the RL agent learns faster than the full model in the clearing blocks task, but the full model eventually outperforms. This is because the full model discovers a novel strategy, different from the strategy employed by human operators. In this case, imitation gave contradictory signals but eventually, reinforcement learning guided the policy towards a better strategy. Second, pouring liquid is the only task where GAIL outperforms its RL counterpart. Imitation can effectively shape the agent's behaviors towards the demonstration trajectories [252]. This is a viable solution for the pouring task, where a controller that generates similar-looking behaviors can complete the task. In contact-rich domains with sufficient variation, however, a controller trained only from a small number of demonstrations will struggle to handle the complex dynamics and to generalize appropriately to novel instances of the task. We hypothesize that this is why the baseline RL agent outperforms the GAIL agent in the other five tasks.

We further perform an ablation study on the block stacking task to understand the impact of different components of our model. In Figure 2.5a, we trained our agents with a number of configurations, each with a single modification to the full model. We see that these ablations cluster into two groups: agents that learn to stack (with average returns greater than 400) and agents that only learn to lift (with average returns between 200 and 300). These results indicate that the hybrid RL/IL reward, learning value function from states, and object-centered features for the discriminator play an integral role in learning good policies. Using only the RL or GAIL reward, learning the

value function from pixels, or providing the full arm state as discriminator input (no discriminator mask) all result in inferior performance. In contrast, the optional components include the recurrent policy core (LSTM), the use of state prediction auxiliary tasks, and whether to include actions in discriminator input. This result suggests that our model can learn end-to-end visuomotor policies without a pretraining phase or the need of auxiliary tasks, as opposed to previous work on visuomotor learning [32, 140, 262]. Furthermore, it can work when the GAIL discriminator only has access to the demonstration states without the accompanying demonstrator actions. Therefore, it can potentially use demonstrations collected with a different body where the underlying controls are unknown or different from the robot's actuators. We then examine the model's sensitivity to the λ values in Equation (2.2). We see in Figure 2.5b that, our model works well with a broad range of λ values from 0.3 to 0.7 that provide a balanced mix of the RL and GAIL rewards.

2.4.4 Sim2Real Policy Transfer Results

To assess the robustness of the simulation-trained policy, we evaluate zero-shot transfer (no additional training) on a real Jaco arm. The real-world setup was roughly matched to the simulation environment (including camera positions and robot kinematics and approximate object size and color). We execute the trained policy network on the robot and count the number of successful trials for both the lifting and stacking tasks. The arm position is randomly initialized and the target block(s) are placed in a number of repeatable start configurations for each task. The zero-shot transfer of the lifting policy has a success rate of 64% over 25 trials (split between 5 block configurations). The stacking policy has a success rate of 35% over 20 trials (split between 2 block configurations). 80% of the stacking trajectories, however, contain successful lifting behavior, and 100% contains successful reaching behavior. It is impractical to conduct a fair comparison with previous work [103, 236, 246] that implemented different tasks and different configurations. The state-of-the-art sim2real work closest to our setup is progressive network [204, 205], which has demonstrated block reaching behaviors with a pixel-to-action RL policy on a Jaco arm. Their work did not demonstrate any lifting or stacking behavior, while our method has achieved reaching behaviors with a 100% success rate. Qualitatively, the policies are notably robust even on failed attempts. The stacking policy repeatedly chases the block to get a successful grasp before trying to stack. For more detailed descriptions of the sim2real results, refer to Appendix A.2.

Several aspects of system mismatch have constrained the policies from attaining a better performance on the real robot. Although the sim and real domains are similar, there is still a sizable reality gap that makes zero-shot transfer challenging. For example, while the simulated blocks are rigid

the objects employed in the real-world setup are non-rigid foam blocks which deform and bounce unpredictably. Furthermore, neural network policies are sensitive to subtle discrepancies between simulated rendering and the real camera frame. Nonetheless, the preliminary successes achieved by these policies offer a good starting point for future work to leverage a small amount of real-world experience to enable better transfer.

2.5 Discussions

In this chapter, we have described a general model-free deep reinforcement learning method for end-to-end learning of policies that operate from RGB camera images and perform manipulation using joint velocity control. Our method combines the use of demonstrations via generative adversarial imitation learning [95] with model-free RL to achieve both effective learning of difficult tasks and robust generalization. The approach only requires a small number of demonstration trajectories (30 per task in the experiments). Additionally, this approach works from state trajectories (without demonstrator actions) combined with the use of only partial/featurized demonstrations being seen by the discriminator – this can simplify and increase the flexibility during data collection and facilitate generalization beyond conditions seen in the demonstrations (e.g., demonstrations could potentially be collected with a different body, such as a human demonstrator via motion capture). Demonstrations were collected via teleoperation of the simulated arm in less than thirty minutes per task. Our method integrates several new techniques to leverage the flexibility and scalability afforded by simulation, such as access to privileged information and the use of large-scale RL algorithms. The experimental results have demonstrated its effectiveness in complex manipulation tasks in simulation and achieved preliminary successes of zero-shot transfer to real hardware. We trained all the policies with the same policy network, the same training algorithm, and the same hyperparameters. The approach makes some use of task-specific information especially in the choice of the object-centric features for the discriminator and the RL reward. In practice we have found the specification of these features intuitive, and our method was reasonably robust to specific choices, thus striking favorable balance between the need for (limited) prior knowledge and the generality of the solutions that can be learned for complex tasks.

In order to fulfill the potential of deep RL in robotics, it is essential to confront the full variability of the real-world, including diversity of object appearances, system dynamics, task semantics, etc. We have therefore focused on learning controllers that could handle significant task variations along multiple dimensions. To improve a policy's ability to generalize, we have increased the diversity of

training conditions with parameterized, procedurally generated 3D objects and randomized system dynamics. This has resulted in policies that exhibit robustness to large variations in simulation as well as against some of the domain discrepancy between simulation and the real world.

Simulation is at the center of our method. Training in simulation circumvents several practical challenges of deep RL for robotics, such as access to state information for reward specification, high sample complexity, and safety considerations. Training in simulation also allows us to use the simulation state to facilitate and stabilize training (i.e. by providing state information to the value function), which in our experiments has been important for learning good visuomotor policies. However, even though our method utilizes such privileged information during training it ultimately produces policies that only rely on vision and proprioceptive information of the arm and that can thus be deployed on real hardware.

Executing the policies on the real robot reveals that there remains a sizable domain gap between simulation and real hardware. Transfer is affected by visual discrepancies as well as by differences in the arm dynamics and in the physical properties of the environment. This leads to a certain level of performance degradation when running a simulation policy on the real robot. Still, our real-world experiments have exemplified that zero-shot sim2real transfer can achieve initial success with RL trained policies performing pixel-to-joint-velocity control.

Chapter 3

Multimodal Representation Learning

3.1 Introduction

Even in routine tasks such as inserting a car key into the ignition, humans effortlessly combine the senses of vision and touch to complete the task. Visual feedback provides semantic and geometric object properties for accurate reaching or grasp pre-shaping. Haptic feedback provides observations of current contact conditions between object and environment for accurate localization and control under occlusions. These two feedback modalities are complementary and concurrent during contact-rich manipulation [20]. Yet, there are few algorithms that endow robots with a similar ability. While the utility of multimodal data has frequently been shown in robotics [18, 199, 224, 245], the proposed manipulation strategies are often task-specific. While learning-based methods do not require manual task specification, the majority of learned manipulation policies close the control loop around a single modality, often vision [32, 63, 140, 272].

In this chapter, we equip a robot with a policy that leverages multimodal feedback from vision and touch, two modalities with very different dimensions, frequencies, and characteristics. This policy is learned through self-supervision and generalizes over variations of the same contact-rich manipulation task in geometry, configurations, and clearances. It is also robust to external perturbations. Our approach starts with using neural networks to learn a shared representation of haptic and visual sensory data, as well as proprioceptive data. Using a self-supervised learning objective, this network is trained to predict optical flow, whether contact will be made in the next control cycle, and concurrency of visual and haptic data. The training is action-conditional to encourage the encoding of action-related information. The resulting compact representation of the high-dimensional and

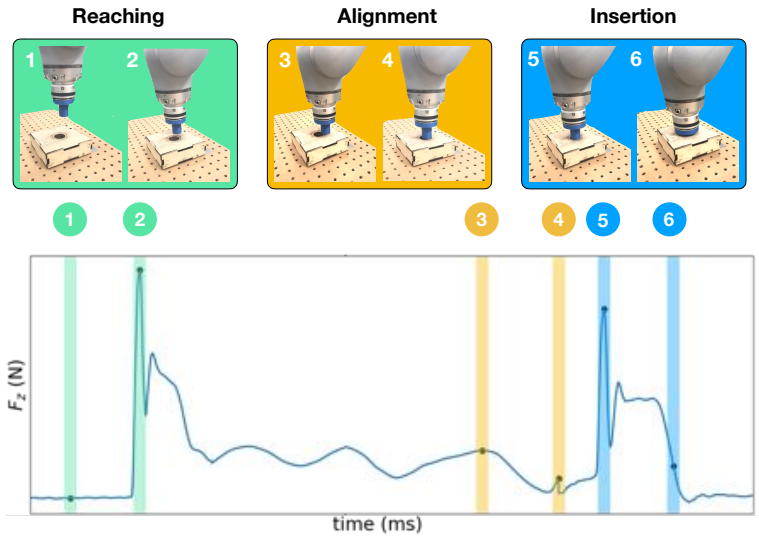


Figure 3.1: Force sensor readings in the z-axis (height) and visual observations are shown with corresponding stages of a peg insertion task. The force reading transitions from (1) the arm moving in free space to (2) making contact with the box. While aligning the peg, the forces capture the sliding contact dynamics on the box surface (3, 4). Finally, in the insertion stage, the forces peak as the robot attempts to insert the peg at the edge of the hole (5), and decrease when the peg slides into the hole (6).

heterogeneous data is the input to a policy for contact-rich manipulation tasks using deep reinforcement learning. The proposed decoupling of state estimation and control achieves practical sample efficiency for learning both representation and policy on a real robot. Our primary contributions are:

- A model for multimodal representation learning from which a contact-rich manipulation policy can be learned.
- Demonstration of insertion tasks that effectively utilize both haptic and visual feedback for hole search, peg alignment, and insertion (see Fig 3.1). Ablative studies compare the effects of each modality on task performance.
- Evaluation of generalization to tasks with different peg geometry and of robustness to perturbation and sensor noise.

3.2 Related Work

3.2.1 Contact-Rich Manipulation

Contact-rich tasks, such as peg insertion, block packing, and edge following, have been studied for decades due to their relevance in manufacturing. Manipulation policies often rely entirely on haptic feedback and force control, and assume sufficiently accurate state estimation [254]. They typically generalize over certain task variations, for instance, peg-in-chamfered-hole insertion policies that work independently of peg diameter [253]. However, entirely new policies are required for new geometries. For chamferless holes, manually defining a small set of viable contact configurations has been successful [29] but cannot accommodate the vast range of real-world variations. [224] combines visual and haptic data for inserting two planar pegs with more complex cross sections, but assumes known peg geometry.

Reinforcement learning approaches have recently been proposed to address variations in geometry and configuration for manipulation. [140, 272] trained neural network policies using RGB images and proprioceptive feedback. Their approach works well in a wide range of tasks, but the large object clearances compared to automation tasks may explain the sufficiency of RGB data. A series of learning-based approaches have relied on haptic feedback for manipulation. Many of them are concerned with estimating the stability of a grasp before lifting an object [16, 31], even suggesting a regrasp [228]. Only a few approaches learn entire manipulation policies through reinforcement only given haptic feedback [110, 229, 242, 243]. While [110] relies on raw force-torque feedback, [229, 242] learn a low-dimensional representation of high-dimensional tactile data before learning a policy. Even fewer approaches exploit the complementary nature of vision and touch. Some of them extend their previous work on grasp stability estimation [15, 30]. Others perform full manipulation tasks based on multiple input modalities [111] but require a pre-specified manipulation graph and demonstrate only on a single task, or require human demonstration and object CAD models [2]. There have been promising works that train manipulation policy in simulation and transfer them to a real robot [9, 26, 182]. However, only few works focused on contact-rich tasks [68] and none relied on haptic feedback in simulation, most likely because of the lack of fidelity of contact simulation and collision modeling for articulated rigid-body systems [59, 70].

3.2.2 Multimodal Representation Learning

The complementary nature of heterogeneous sensor modalities has previously been explored for inference and decision making. The diverse set of modalities includes vision, range, audio, haptic and proprioceptive data as well as language. This heterogeneous data makes the application of hand-designed features and sensor fusion extremely challenging. That is why learning-based methods have been on the forefront. [15, 30, 71, 223] are examples of fusing visual and haptic data for grasp stability assessment, manipulation, material recognition, or object categorization. [149, 229] fuse vision and range sensing and [229] adds language labels. While many of these multimodal approaches are trained through a classification objective [15, 30, 71, 263], we are interested in multimodal representation learning for control. A popular representation learning objective is reconstruction of the raw sensory input [44, 139, 242, 263]. This unsupervised objective benefits learning stability and speed, but it is also data intensive and prone to overfitting [44]. When learning for control, action-conditional predictive representations can encourage the state representations to capture action-relevant information [139]. Studies attempted to predict full images when pushing objects with benign success [3, 11, 172]. In these cases either the underlying dynamics is deterministic [172], or the control runs at a low frequency [63]. In contrast, we operate with haptic feedback at 1kHz and send Cartesian control commands at 20Hz. We use an action-conditional surrogate objective for predicting optical flow and contact events with self-supervision.

There is compelling evidence that the interdependence and concurrency of different sensory streams aid perception and manipulation [21, 54, 132]. However, few studies have explicitly exploited this concurrency in representation learning. Examples include [225] for visual prediction tasks and [168, 175] for audio-visual coupling. Following [175], we propose a self-supervised objective to fuse visual and haptic data.

3.3 Problem Formulation

Our goal is to learn a policy on a robot for performing contact-rich manipulation tasks. We want to evaluate the value of combining multisensory information and the ability to transfer multimodal representations across tasks. For sample efficiency, we first learn a neural network-based feature representation of the multisensory data. The resulting compact feature vector serves as input to a policy that is learned through reinforcement learning.

We model the manipulation task as a finite-horizon, discounted Markov Decision Process (MDP) $\mathscr{M}$, with a state space $\mathscr{S}$, an action space $\mathscr{A}$, state transition dynamics $\mathscr{T} : \mathscr{S} \times \mathscr{A} \to \mathscr{S}$, an initial

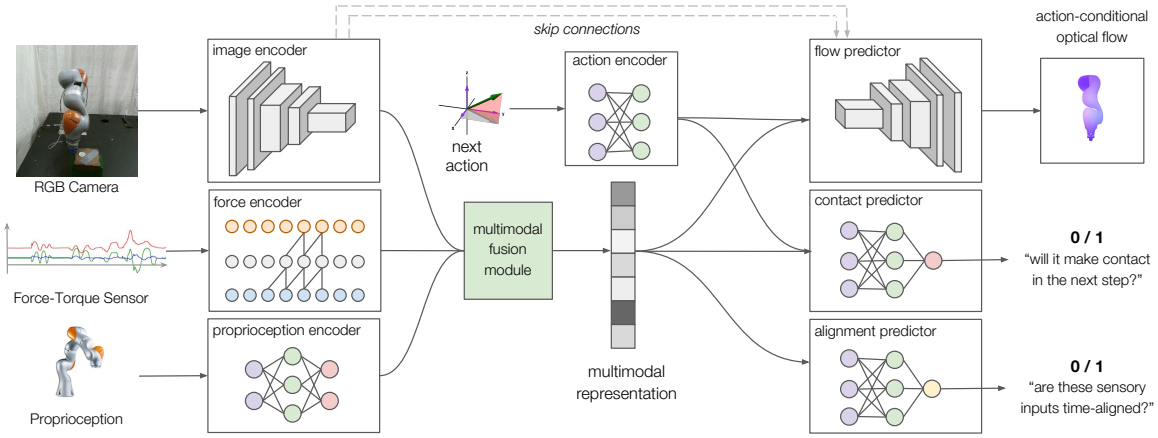


Figure 3.2: Neural network architecture for multimodal representation learning with self-supervision. The network takes data from three different sensors as input: RGB images, F/T readings over a 32ms window, and end-effector position and velocity. It encodes and fuses this data into a multimodal representation based on which controllers for contact-rich manipulation can be learned. This representation learning network is trained end-to-end through self-supervision.

state distribution ρ_0, a reward function $r : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$, horizon T, and discount factor $\gamma \in (0, 1]$. To determine the optimal stochastic policy $\pi : \mathscr{S} \to \mathbb{P}(\mathscr{A})$, we want to maximize the expected discounted reward

$$J(\pi) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T-1} \gamma r(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{3.1}$$

We represent the policy by a neural network with parameters θ_π that are learned as described in Section 3.5. $\mathscr{S}$ is defined by the low-dimensional representation learned from high-dimensional visual and haptic sensory data. This representation is a neural network parameterized by θ_s and is trained as described in Section 3.4. $\mathscr{A}$ is defined over continuously-valued, 3D displacements $\Delta\mathbf{x}$ in Cartesian space. The controller design is detailed in Section 3.5.

3.4 Model

Deep networks are powerful tools to learn representations from high-dimensional data [135] but require a substantial amount of training data. Here, we address the challenge of seeking sources of supervision that do not rely on laborious human annotation. We design a set of predictive tasks that are suitable for learning visual and haptic representations for contact-rich manipulation tasks, where supervision can be obtained via automatic procedures rather than manual labeling. Figure 3.2

visualizes our representation learning model.

3.4.1 Modality Encoders

Our model encodes three types of sensory data available to the robot: RGB images from a fixed camera, haptic feedback from a wrist-mounted force-torque (F/T) sensor, and proprioceptive data from the joint encoders of the robot arm. The heterogeneous nature of this data requires domain-specific encoders to capture the unique characteristics of each modality. For visual feedback, we use a 6-layer *convolutional neural network* (CNN) similar to FlowNet [66] to encode $128 \times 128 \times 3$ RGB images. We add a fully-connected layer to transform the final activation maps into a 128-d feature vector. For haptic feedback, we take the last 32 readings from the six-axis F/T sensor as a 32×6 time series and perform 5-layer causal convolutions [173] with stride 2 to transform the force readings into a 64-d feature vector. For proprioception, we encode the current position and velocity of the end-effector with a 2-layer *multilayer perceptron* (MLP) to produce a 32-d feature vector. The resulting three feature vectors are concatenated into one vector and passed through the multimodal fusion module (2-layer MLP) to produce the final 128-d multimodal representation.

3.4.2 Self-Supervised Predictions

The modality encoders have nearly half a million learnable parameters and require a large amount of labeled training data. To avoid manual annotation, we design training objectives for which labels can be automatically generated through self-supervision. Furthermore, representations for control should encode the action-related information. To achieve this, we design two action-conditional representation learning objectives. Given the next robot action and the compact representation of the current sensory data, the model has to predict (i) the optical flow generated by the action and (ii) whether the end-effector will make contact with the environment in the next control cycle. Ground-truth optical flow annotations are automatically generated given proprioception and known robot kinematics and geometry [66, 73]. Ground-truth annotations of binary contact states are generated by applying simple heuristics on the F/T readings.

The next action, i.e. the end-effector motion, is encoded by a 2-layer MLP. Together with the multimodal representation it forms the input to the flow and contact predictor. The flow predictor uses a 6-layer convolutional decoder with upsampling to produce a flow map of size $128 \times 128 \times 2$. Following [66], we use 4 skip connections. The contact predictor is a 2-layer MLP and performs binary classification.

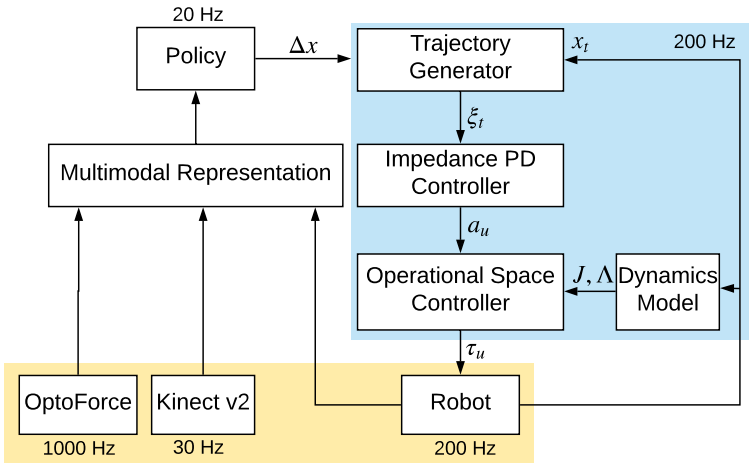


Figure 3.3: Our controller takes end-effector position displacements from the policy at 20Hz and outputs robot torque commands at 200Hz. The trajectory generator interpolates high-bandwidth robot trajectories from low-bandwidth policy actions. The impedance PD controller tracks the interpolated trajectory. The operational space controller uses the robot dynamics model to transform Cartesian-space accelerations into commanded joint torques. The resulting controller is compliant and reactive.

As discussed in Section 3.2.2, there is concurrency between the different sensory streams leading to correlations and redundancy, e.g., seeing the peg, touching the box, and feeling the force. We exploit this by introducing a third representation learning objective that predicts whether two sensor streams are temporally aligned [175]. During training, we sample a mix of time-aligned multimodal data and randomly shifted ones. The alignment predictor (a 2-layer MLP) takes the low-dimensional representation as input and performs binary classification of whether the input was aligned or not.

We train the action-conditional optical flow with endpoint error (EPE) loss averaged over all pixels [66], and both the contact prediction and the alignment prediction with cross-entropy loss. During training, we minimize a sum of the three losses end-to-end with stochastic gradient descent on a dataset of rolled-out trajectories. Once trained, this network produces a 128-d feature vector that compactly represents multimodal data. This vector from the input to the manipulation policy learned via reinforcement learning.

3.5 Policy Learning and Controller Design

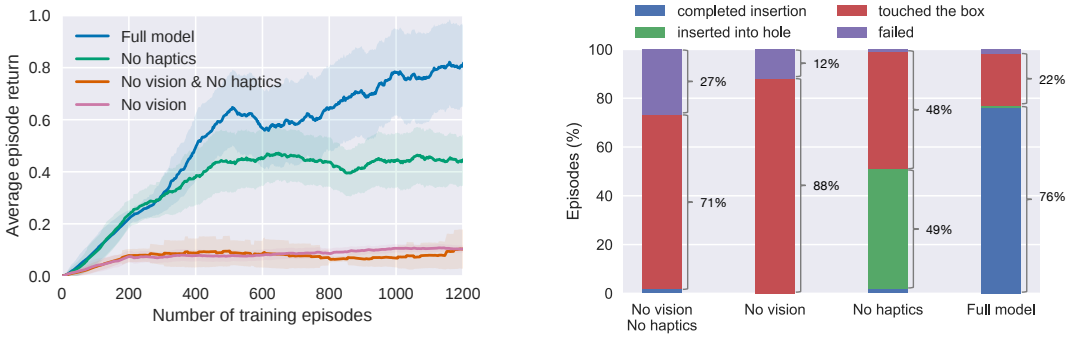
Our final goal is to equip a robot with a policy for performing contact-rich manipulation tasks that leverage multimodal feedback. Though it is possible to engineer controllers for specific instances

of these tasks [224, 254], this effort is difficult to scale to the large variability of real-world tasks. Therefore, it is desirable to enable a robot to supervise itself where the learning process is applicable to a broad range of tasks. Given its recent success in continuous control [146, 213], deep reinforcement learning lends itself well to learning policies that map high-dimensional features to control commands.

Policy Learning. Modeling contact interactions and multi-contact planning still result in complex optimization problems [191, 189, 239] that remain sensitive to inaccurate actuation and state estimation. We formulate contact-rich manipulation as a model-free reinforcement learning problem to investigate its performance when relying on multimodal feedback and when acting under uncertainty in geometry, clearance and configuration. By choosing model-free, we also eliminate the need for an accurate dynamics model, which is typically difficult to obtain in the presence of rich contacts. Specifically, we choose trust-region policy optimization (TRPO) [213]. TRPO imposes a bound of KL-divergence for each policy update by solving a constrained optimization problem, which prevents the policy from moving too far away from the previous step. The policy network is a 2-layer MLP that takes as input the 128-d multimodal representation and produces a 3D displacement $\Delta\mathbf{x}$ of the robot end-effector. To train the policy efficiently, we freeze the representation model parameters during policy learning, such that it reduces the number of learnable parameters to 3% of the entire model and substantially improves the sample efficiency.

Controller Design. Our controller takes as input Cartesian end-effector displacements $\Delta\mathbf{x}$ from the policy at 20Hz, and outputs direct torque commands τ_{u} to the robot at 200Hz. Its architecture can be split into three parts: trajectory generation, impedance control and operational space control (see Fig 3.3). Our policy outputs Cartesian control commands instead of joint-space commands, so it does not need to implicitly learn the non-linear and redundant mapping between 7-DoF joint space and 3-DoF Cartesian space. We use direct torque control as it gives our robot compliance during contact, which makes the robot safer to itself, its environment, and any nearby human operator. In addition, compliance makes the peg insertion task easier to accomplish under position uncertainty, as the robot can slide on the surface of the box while pushing downwards [55, 110, 198].

The trajectory generator bridges low-bandwidth output of the policy (limited by the forward pass of our representation model), and the high-bandwidth torque control of the robot. Given $\Delta\mathbf{x}$ from the policy and the current end-effector position $\mathbf{x}_t$, we calculate the desired end-effector position $\mathbf{x}_{\text{des}}$. The trajectory generator interpolates between $\mathbf{x}_t$ and $\mathbf{x}_{\text{des}}$ to yield a trajectory $\xi_t = \{\mathbf{x}_k, \mathbf{v}_k, \mathbf{a}_k\}_{k=t}^{t+T}$ of end-effector position, velocity and acceleration at 200Hz. This forms the input to a PD impedance controller to compute a task space acceleration command: $\mathbf{a}_u = \mathbf{a}_{\text{des}} - \mathbf{k}_{\text{p}}(\mathbf{x} - \mathbf{x}_{\text{des}}) - \mathbf{k}_{\text{v}}(\mathbf{v} - \mathbf{v}_{\text{des}})$,



(a) Training curves of reinforcement learning

(b) Policy evaluation statistics

Figure 3.4: Simulated peg insertion. Ablative study of representations trained on different combinations of sensory modalities. We compare our full model, trained with a combination of visual and haptic feedback and proprioception, with baselines that are trained without vision, or haptics, or either. (b) The graph shows partial task completion rates with different feedback modalities, and we note that both the visual and haptic modalities play an integral role for contact-rich tasks.

where $\mathbf{k}_\text{p}$ and $\mathbf{k}_\text{v}$ are manually tuned gains.

By leveraging known kinematic and dynamics models of the robot, we can calculate joint torques from Cartesian space accelerations with the dynamically-consistent operational space formulation [113]. We compute the force at the end-effector with $\mathbf{F} = \Lambda \mathbf{a}_u$, where Λ is the inertial matrix in the end-effector frame that decouples the end-effector motions. Finally, we map from $\mathbf{F}$ to joint torque commands with the end-effector Jacobian J, which is a function of joint angle $\mathbf{q}$: $\tau_\text{u} = J^T(\mathbf{q})\mathbf{F}$.

3.6 Experiments

The primary goal of our experiments is to examine the effectiveness of the multimodal representations in contact-rich manipulation tasks. In particular, we design the experiments to answer the following three questions: 1) What is the value of using *all* instead of a subset of modalities? 2) Is policy learning on the real robot *practical* with a learned representation? 3) Does the learned representation *generalize* over task variations and recover from perturbations?

Task Setup. We design a set of peg insertion tasks where task success requires joint reasoning over visual and haptic feedback. We use five different types of pegs and holes fabricated with a 3D printer: round peg, square peg, triangular peg, semicircular peg, and hexagonal peg, each with a nominal clearance of around 2mm as shown in Figure 3.5a.

Robot Environment Setup. For both simulation and real robot experiments, we use the Kuka

LBR IIWA robot, a 7-DoF torque-controlled robot. Three sensor modalities are available in both simulation and real hardware, including proprioception, an RGB camera, and a force-torque sensor. The proprioceptive feature is the end-effector pose as well as linear and angular velocity. They are computed using forward kinematics. RGB images are recorded from a fixed camera pointed at the robot. Input images to our model are down-sampled to 128×128. On the real robot, we use the Kinect v2 camera. In simulation, we use CHAI3D [36] for rendering. The force sensor provides a 6-axis feedback on the forces and moments along the x, y, z axes. On the real robot, we mount an OptoForce sensor between the last joint and the peg. In simulation, the contact between the peg and the box is modeled with SAI 2.0 [37], a real-time physics simulator for rigid articulated bodies with high fidelity contact resolution.

Reward Design. We use the following staged reward function to guide the reinforcement learning algorithm through the different sub-tasks, simplifying the challenge of exploration and improving learning efficiency:

$$r(\mathbf{s}) = \begin{cases} c_r - \frac{c_r}{2}(\tanh \lambda \|\mathbf{s}\| + \tanh \lambda \|\mathbf{s}_{xy}\|) & \text{(reaching)} \\ 2 - c_a \|\mathbf{s}_{xy}\|_2 & \text{if } \|\mathbf{s}_{xy}\|_2 \leq \varepsilon_1 \quad \text{(alignment)} \\ 4 - 2(\frac{s_z}{h_d - \varepsilon_2}) & \text{if } s_z < 0 \quad \text{(insertion)} \\ 10 & \text{if } h_d - |s_z| \leq \varepsilon_2 \quad \text{(completion)}, \end{cases}$$

where $\mathbf{s} = (s_x, s_y, s_z)$ and $\mathbf{s}_{xy} = (s_x, s_y)$ use the peg's current position, λ is a constant factor to scale the input to the tanh function. The target peg position is $(0, 0, -h_d)$ with h_d as the height of the hole, and c_r and c_a are constant scale factors.

Evaluation Metrics. We report the quantitative performance of the policies using the sum of rewards achieved in an episode, normalized by the highest attainable reward. We also provide the statistics of the stages of the peg insertion task that each policy can achieve, and report the percentage of evaluation episodes in the following four categories:

1. *completed insertion*: the peg reaches bottom of the hole;
2. *inserted into hole*: the peg goes into the hole but has not reached the bottom;
3. *touched the box*: the peg only makes contact with the box;
4. *failed*: the peg fails to reach the box.

Implementation Details. To train each representation model, we collect a multimodal dataset of 100k states and generate the self-supervised annotations. We roll out a random policy as well as a

heuristic policy while collecting the data, which encourages the peg to make contact with the box. As the policy runs at 20 Hz, it takes 90 to 120 minutes to collect the data. The representation models are trained for 20 epochs on a Titan V GPU before starting policy learning.

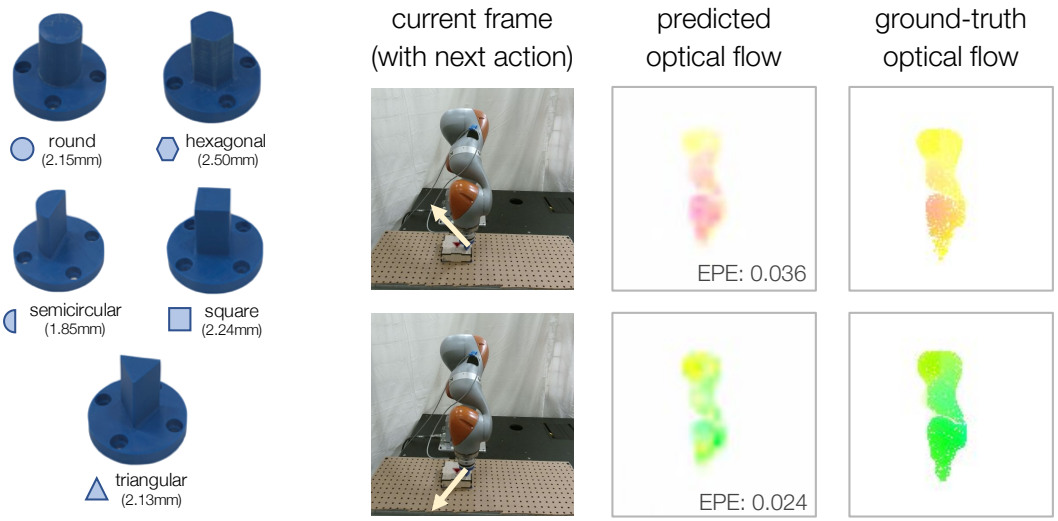
We first conduct an ablative study in simulation to investigate the contributions of individual sensory modalities to learning the multimodal representation and manipulation policy. We then apply our full multimodal model to a real robot, and train reinforcement learning policies for the peg insertion tasks from the learned representations with high sample efficiency. Furthermore, we visualize the representations and provide a detailed analysis of robustness with respect to shape and clearance variations.

3.6.1 Simulation Experiments

Three modalities are encoded and fused by our representation model: RGB images, force readings, and proprioception (see Figure 3.2). To investigate the importance of each modality for contact-rich manipulation tasks, we perform an ablative study in simulation, where we learn the multimodal representations with different combinations of modalities. These learned representations are subsequently fed to the TRPO policies to train on a task of inserting a square peg. We randomize the configuration of the box position and the arm's initial position at the beginning of each episode to enhance the robustness and generalization of the model.

We illustrate the training curves of the TRPO agents in Figure 3.4a. We train all policies with 1.2k episodes, each lasting 500 steps. We evaluate 10 trials with the stochastic policy every 10 training episodes and report the mean and standard deviation of the episode rewards. Our `Full model` corresponds to the multimodal representation model introduced in Section 3.4, which takes all three modalities as input. We compare it with three baselines: `No vision` masks out the visual input to the network, `No haptics` masks out the haptic input, and `No vision No haptics` leaves only proprioceptive input. From Figure 3.4a we observe that the absence of either the visual or force modality negatively affects task completion, with `No vision No haptics` performing the worst. None of the three baselines has reached the same level of performance as the final model. Among these three baselines, we see that the `No haptics` baseline achieved the highest rewards. We hypothesize that vision locates the box and the hole, which facilitates the first steps of robot reaching and peg alignment, while haptic feedback is uninformative until after contact is made.

The `Full model` achieves the highest success rate with nearly 80% completion rate, while all baseline methods have a completion rate below 5%. It is followed by the `No haptics` baseline,



(a) Peg variations

(b) Optical flow prediction examples

Figure 3.5: (a) 3D printed pegs used in the real robot experiments and their box clearances. (b) Qualitative predictions: We visualize examples of optical flow predictions from our representation model (using color scheme in [66]). The model predicts different flow maps on the same image conditioned on different next actions indicated by projected arrows.

which relies solely on the visual feedback. We see that it is able to localize the hole and perform insertion half of the time from only the visual inputs; however, few episodes have completed the full insertion. It implies that the haptic feedback plays a more crucial role in determining the actions when the peg is placed in the hole. The remaining two baselines can often reach the box through random exploration, but are unable to exhibit consistent insertion behaviors.

3.6.2 Real Robot Experiments

We evaluate our `Full model` on the real hardware with round, triangular, and semicircular pegs. In contrast to simulation, the difficulty of sensor synchronization, variable delays from sensing to control, and complex real-world dynamics introduce additional challenges on the real robot. We make the task tractable on a real robot by training a shallow neural network controller while freezing the multimodal representation model that can generate action-conditional flows with low endpoint errors (see Figure 3.5b).

We train the TRPO policies for 300 episodes, each lasting 1000 steps, roughly 5 hours of wall-clock time. We evaluate each policy for 100 episodes in Figure 3.6. The first three bars correspond to the set of experiments where we train a specific representation model and policy for each type of

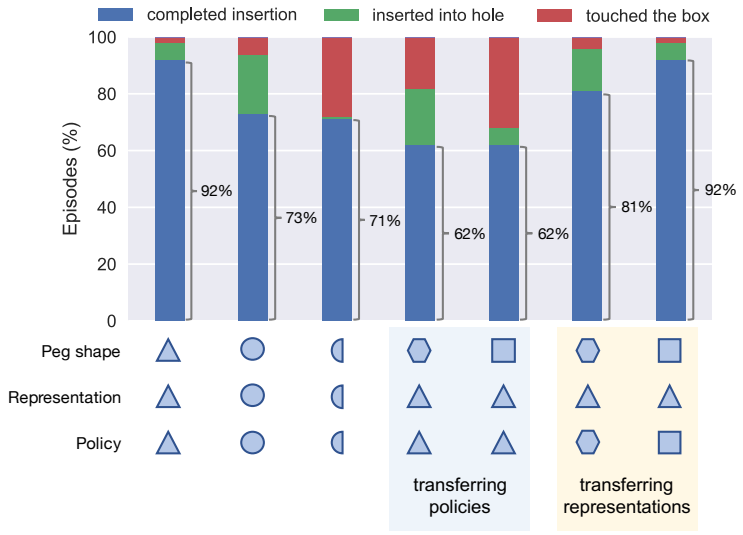


Figure 3.6: Real robot peg insertion. We evaluate our `Full Model` on the real hardware with different peg shapes, indicated on the x-axis. The learned policies achieve the tasks with a high success rate. We also study transferring the policies and representations from trained pegs to novel peg shapes (last four bars). The robot effectively re-uses previously trained models to solve new tasks.

peg. The robot achieves a level of success similar to that in simulation. A common strategy that the robot learns is to reach the box, search for the hole by sliding over the surface, align the peg with the hole, and finally perform insertion.

We further examine the potential of transferring the learned policies and representations to two novel shapes previously unseen in representation and policy training, the hexagonal peg and the square peg. For policy transfer, we take the representation model and the policy trained for the triangular peg, and execute with the new pegs. From the 4th and 5th bars in Figure 3.6, we see that the policy achieves over 60% success rate on both pegs without any further policy training on them. A better transfer performance can be achieved by taking the representation model trained on the triangular peg, and training a new policy for the new pegs. As shown in the last two bars in Figure 3.6, the resulting performance increases 19% for the hexagonal peg and 30% for the square peg. Our transfer learning results indicate that the multimodal representations from visual and haptic feedback generalize well across variations of our contact-rich manipulation tasks.

Finally, we study the robustness of our policy in the presence of sensory noise and external perturbations to the arm by periodically occluding the camera and pushing the robot arm during trajectory roll-out. The policy is able to recover from both the occlusion and perturbations.

3.7 Discussions

In this chapter, we examined the value of jointly reasoning over time-aligned multi-sensory data for contact-rich manipulation tasks. To enable efficient real robot training, we proposed a novel model to encode heterogeneous sensory inputs into a compact multimodal representation. Once trained, the representation remained fixed when being used as input to a shallow neural network policy for reinforcement learning. We trained the representation model with self-supervision, eliminating the need for manual annotation. Our experiments with tight clearance peg insertion tasks indicated that they require the multimodal feedback from both vision and touch. We further demonstrated that the multimodal representations transfer well to new task instances of peg insertion. For future work, we plan to extend our method to other contact-rich tasks, which require a full 6-DoF controller of position and orientation. We would also like to explore the value of incorporating richer modalities, such as depth and sound, into our representation learning pipeline, as well as new sources of self-supervision.

Chapter 4

Target-Driven Visual Navigation

4.1 Introduction

Many tasks in robotics involve interactions with physical environments and objects. One of the fundamental components of such interactions is understanding the correlation and causality between actions of an agent and the changes of the environment as a result of the action. Since the 1970s, there have been various attempts to build a system that can understand such relationships. Recently, with the rise of deep learning models, learning-based approaches have gained wide popularity [140, 161].

In this chapter, we focus on the problem of navigating a space to find a given target using only visual input. Successful navigation requires learning relationships between actions and the environment. This makes the task well suited to a Deep Reinforcement Learning (DRL) approach. However, general DRL approaches (e.g., [160, 161]) are designed to learn a policy that depends only on the current state, and the goal is implicitly embedded in the model parameters. Hence, it is necessary to learn new model parameters for a new target. This is problematic since training DRL agents is computationally expensive.

In order to achieve higher *adaptability* and *flexibility*, we introduce a *target-driven model*. Our model takes the visual task objective as an input. Thus we can avoid re-training for every new target. Our model learns a policy that jointly embeds the target and the current state. Essentially, an agent learns to take its next action conditioned on both its current state and target, rather than its current state only. Hence, there is no need to re-train the model for new targets. A key intuition that we rely on is that different training episodes share information. For example, agents explore common routes during the training stage while being trained for finding different targets. Various scenes also

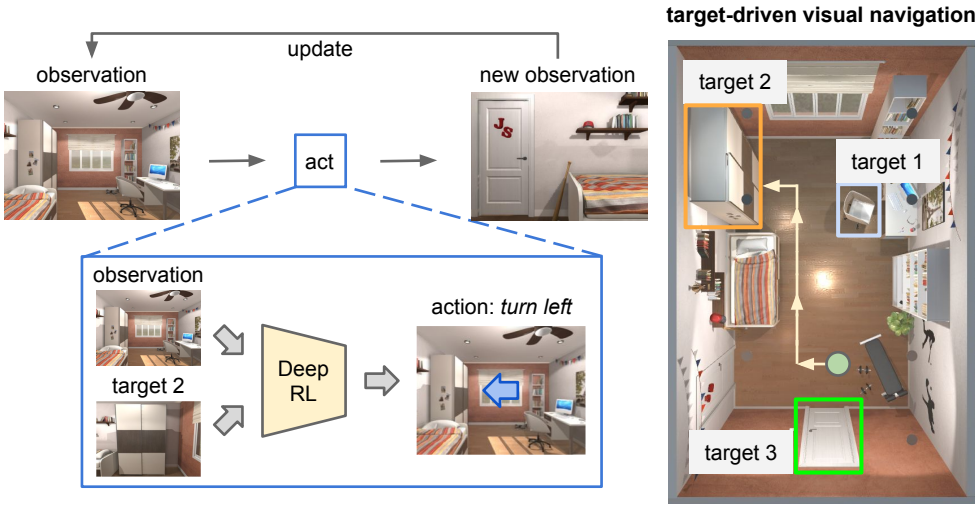


Figure 4.1: The goal of our deep reinforcement learning model is to navigate towards a visual target with a minimum number of steps. Our model takes the current observation and the image of the target as input and generates an action in the 3D environment as the output. Our model learns to navigate to different targets in a scene without re-training.

share similar structures and statistics (e.g., a fridge is likely to be near a microwave). In short, we exploit the fact that learning for new targets will be easier with the models that have been trained for other targets.

Unfortunately, training and quantitatively evaluating DRL algorithms in real environments is often tedious. One reason is that running systems in a physical space is time consuming. Furthermore, acquiring large-scale action and interaction data in real environments is not trivial via the common image dataset collection techniques. To this end, we developed one of the first simulation frameworks with high-quality 3D scenes, called The House Of inteRactions (AI2-THOR). Our simulation framework enables us to collect a large number of visual observations for action and reaction in different environments. For example, an agent can freely navigate (i.e. move and rotate) in various realistic indoor scenes, and is able to have low- and high-level interactions with the objects (e.g., applying a force or opening/closing a microwave).

We evaluate our method for the following tasks: (1) *Target generalization*, where the goal is to navigate to targets that have not been used during training within a scene. (2) *Scene generalization*, where the goal is to navigate to targets in scenes not used for training. (3) *Real-world generalization*, where we demonstrate navigation to targets using a real robot. Our experiments show that we outperform the state-of-the-art DRL methods in terms of data efficiency for training. We also

demonstrate the generalization aspects of our model.

In summary, we introduce a novel reinforcement learning model that generalizes across targets and scenes. To learn and evaluate reinforcement learning models, we create a simulation framework with high-quality rendering that enables visual interactions for agents. We also demonstrate real robot navigation using our model generalized to the real world with a small amount of fine-tuning.

4.2 Related Work

There is a large body of work on *visual* navigation. We provide a brief overview of some of the relevant work. The map-based navigation methods require a global map of the environment to make decisions for navigation (e.g., [23, 24, 115, 174]). One of the main advantages of our method over these approaches is that it does not need a prior map of the environment. Another class of navigation methods reconstruct a map on the fly and use it for navigation [40, 42, 221, 238, 256], or go through a training phase guided by humans to build the map [114, 202]. In contrast, our method does not require a map of the environment, as it does not have any assumption on the landmarks of the environment, nor does it require a human-guided training phase. Map-less navigation methods are common as well [43, 86, 137, 195, 209]. These methods mainly focus on obstacle avoidance given the input image. Our method is considered *map-less*. However, it possesses implicit knowledge of the environment. A survey of visual navigation methods can be found in [22].

Note that our approach is not based on feature matching or 3D reconstruction, unlike, e.g., [123, 184]. Besides, our method does not require supervised training for recognizing distinctive landmarks, unlike, e.g., [147, 157].

Reinforcement Learning (RL) has been used in a variety of applications. [120] propose a policy gradient RL approach for locomotion of a four-legged robot. [183] discuss policy gradient methods for learning motor primitives. [159] propose an RL-based method for obstacle detection using a monocular camera. [116] apply reinforcement learning to autonomous helicopter flight. [121] use RL to automate data collection process for mapping. [14] propose a kernel-based reinforcement learning algorithm for large-scale settings. [145] use RL for making decisions in ATARI games. In contrast to these approaches, our models use deep reinforcement learning to handle high-dimensional sensory inputs.

Recently, methods that integrate deep learning methods with RL have shown promising results. [161] propose deep Q-networks to play ATARI games. [219] propose a new search algorithm based on the integration of Monte-Carlo tree search with deep RL that beats the world champion in the

game of Go. [160] propose a deep RL approach, where the parameters of the deep network are updated by multiple asynchronous copies of the agent in the environment. [140] use a deep RL approach to directly map the raw images into torques at robot motors. Our work deals with much more complex inputs than ATARI games, or images taken in a lab setting with a constrained background. Additionally, our method is generalizable to new scenes and new targets, while the mentioned methods should be re-trained for a new game, or in case of a change in the game rules.

There have been some effort to develop learning methods that can generalize to different target tasks [178, 206]. Similar to UVFA [210], our model takes the target goal directly as an input without the need of re-training.

Recently, physics engines have been used to learn the dynamics of real-world scenes from images [162, 163, 257]. In this chapter, we show that a model that is trained in simulation can be generalized to real-world scenarios.

4.3 The AI2-THOR Framework

To train and evaluate our model, we require a framework for performing actions and perceiving their outcomes in a 3D environment. Integrating our model with different types of environments is a main requirement for generalization of our model. Hence, the framework should have a plug-n-play architecture such that different types of scenes can be easily incorporated. Additionally, the framework should have a detailed model of the physics of the scene so the movements and object interactions are properly represented.

For this purpose, we propose The House Of inteRactions (AI2-THOR) framework, which is designed by integrating a physics engine (Unity 3D) with a deep learning framework (Tensorflow [1]). The general idea is that the rendered images of the physics engine are streamed to the deep learning framework, and the deep learning framework issues a control command based on the visual input and sends it back to the agent in the physics engine. Similar frameworks have been proposed by [17, 107, 112, 138, 259], but the main advantages of our framework are as follows: (1) The physics engine and the deep learning framework directly communicate (in contrast to separating the physics engine from the controller as in [163]). Direct communication is important since the feedback from the environment can be immediately used for online decision making. (2) We tried to mimic the appearance distribution of the real-world images as closely as possible. For example, [17] work on Atari games, which are 2D environments and limited in terms of appearance. [88] is a collection of synthetic scenes that are non-photo-realistic and do not follow the distribution of real-world scenes

Figure 4.2: Screenshots of our framework and other simulated learning frameworks: ALE [17], ViZDoom [112], UETorch [138], Project Malmo [107], SceneNet [88], TORCS [259], SYNTHIA [200], Virtual KITTI [69].

in terms of lighting, object appearance, textures, and background clutter, etc. This is important for enabling us to generalize to real-world images.

To create indoor scenes for our framework, we provided reference images to artists to create a 3D scene with the texture and lighting similar to the image. So far we have 32 scenes that belong to 4 common scene types in a household environment: kitchen, living room, bedroom, and bathroom. On average, each scene contains 68 object instances.

The advantage of using a physics engine for modeling the world is that it is highly scalable (training a robot in real houses is not easily scalable). Furthermore, training the models can be performed cheaper and safer (e.g., the actions of the robot might damage objects). One main drawback of using synthetic scenes is that the details of the real world are under-modeled. However, recent advances in the graphics community make it possible to have a rich representation of the real-world appearance and physics, narrowing the discrepancy between real world and simulation. Figure 4.2 provides a qualitative comparison between a scene in our framework and example scenes in other frameworks and datasets. As shown, our scenes better mimic the appearance properties of real world scenes. In this work, we focus on navigation, but the framework can be used for more fine-grained physical interactions, such as applying a force, grasping, or object manipulations such

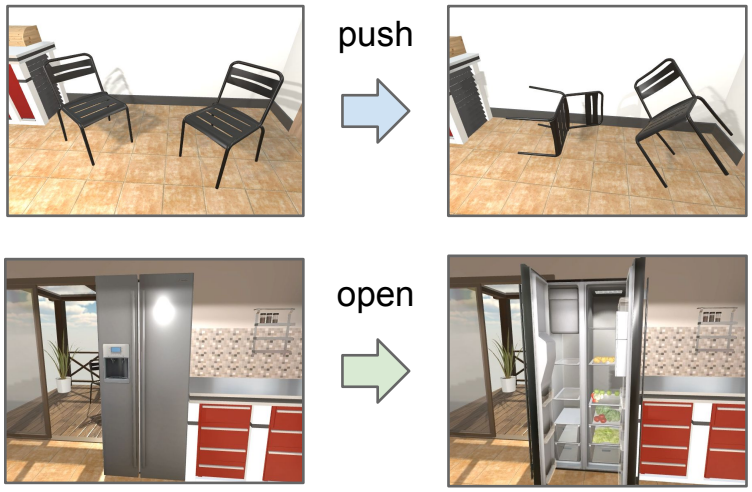


Figure 4.3: Our framework provides a rich interaction platform for AI agents. It enables physical interactions, such as pushing or moving objects (the first row), as well as object interactions, such as changing the state of objects (the second row).

as opening and closing a microwave. Figure 4.3 shows a few examples of high-level interactions. We will provide Python APIs with our framework for an AI agent to interact with the 3D scenes.

4.4 Model

In this section, we first define our formulation for target-driven visual navigation. Then we describe our deep siamese actor-critic network for this task.

4.4.1 Problem Formulation

Our goal is to find the minimum length sequence of actions that move an agent from its current location to a target that is specified by an RGB image. We develop a deep reinforcement learning model that takes as input an RGB image of the current observation and another RGB image of the target. The output of the model is an action in 3D such as move forward or turn right. Note that the model learns a mapping from the 2D image to an action in the 3D space.

Vision-based robot navigation requires a mapping from sensory signals to motion commands. Previous work on Reinforcement Learning typically do not consider high-dimensional perceptual inputs [119]. Recent deep reinforcement learning (DRL) models [161] provide an end-to-end learning framework for transforming pixel information into actions. However, DRL has largely focused

on learning goal-specific models that tackle individual tasks in isolation. This training setup is rather inflexible to changes in task goals. For instance, as pointed out by Lake et al. [133], changing the rule of the game would have devastating performance impact on DRL-based Go-playing systems [219]. Such limitation roots from the fact that standard DRL models [160, 161] aim at finding a direct mapping (represented by a deep neural network π) from state representations s to policy $\pi(s)$. In such cases, the goal is hardcoded in neural network parameters. Thus, changes in goals would require to update the network parameters in accordance.

Such limitation is especially problematic for mobile robot navigation. When applying DRL to the multiple navigation targets, the network should be re-trained for each target. In practice, it is prohibitive to exhaust every target in a scene. This is the problem caused by a lack of generalization – i.e., we would have to re-train a new model when incorporating new targets. Therefore, it is preferable to have a single navigation model, which learns to navigate to new targets without re-training. To achieve this, we specify the task objective (i.e., navigation destination) as inputs to the model, instead of implanting the target in the model parameters. We refer to this problem as *target-driven visual navigation*. Formally, the learning objective of a target-driven model is to learn a stochastic policy function π which takes two inputs, a representation of current state s_t and a representation of target g and produces a probability distribution over the action space $\pi(s_t, g)$. For testing, a mobile robot keeps taking actions drawn from the policy distribution until reaching the destination. This way, actions are conditioned on both states and targets. Hence, no re-training for new targets is required.

4.4.2 Learning Setup

Before introducing our model, we first describe the key ingredients of the reinforcement learning setup: action space, observations and goals, and reward design.

Action space

Real-world mobile robots have to deal with low-level mechanics. However, such mechanical details make the learning significantly more challenging. A common approach is to learn at a certain level of abstraction, where the underlying physics is handled by a lower-level controller (e.g., 3D physical engine). We train our model with command-level actions. For our visual navigation tasks, we consider four actions: moving forward, moving backward, turning left, and turning right. We use a constant step length (0.5 meters) and turning angle (90 degree). This essentially discretizes the

scene space into a *grid-world* representation. To model uncertainty in real-world system dynamics, we add a Gaussian noise to steps $\mathcal{N}(0, 0.01)$ and turns $\mathcal{N}(0, 1.0)$ at each location.

Observations and Goals

Both observations and goals are images taken by the agent's RGB camera in its first-person view. The benefit of using images as goal descriptions is the flexibility for specifying new targets. Given a target image, the task objective is to navigate to the location and viewpoint where the target image is taken.

Reward design

We focus on minimizing the trajectory length to the navigation targets, although other factors such as energy efficiency could be considered. We only provide a goal-reaching reward (10.0) upon task completion. To encourage shorter trajectories, we add a small time penalty (-0.01) as immediate reward.

4.4.3 Target-Driven Navigation Model

We focus on learning the target-driven policy function π via deep reinforcement learning. We design a new deep neural network as a non-linear function approximator for π, where action a at time t can be drawn by:

$$a \sim \pi(s_t, g \,|\, \mathbf{u}) \tag{4.1}$$

where $\mathbf{u}$ are the model parameters, s_t is the image of the current observation, and g is the image of the navigation target. When target g belongs to a finite discrete set, π can be seen as a mixture model, where g indexes the right set of parameters for each goal. However, the number of real-world goals is often countless (due to many different locations or highly variable object appearances). Thus, it is preferable to learn a projection that transforms the goals into an embedding space. Such projection enables knowledge transfer across this embedding space, and therefore allows the model to generalize to new targets.

Navigational decisions demand an understanding of the relative spatial positions between the current locations and the target locations, as well as a holistic sense of scene layout. We develop a new deep siamese actor-critic network to capture such intuitions. Figure 4.4 illustrates our model for the target-driven navigation tasks. Overall, the inputs to the network are two images that represent

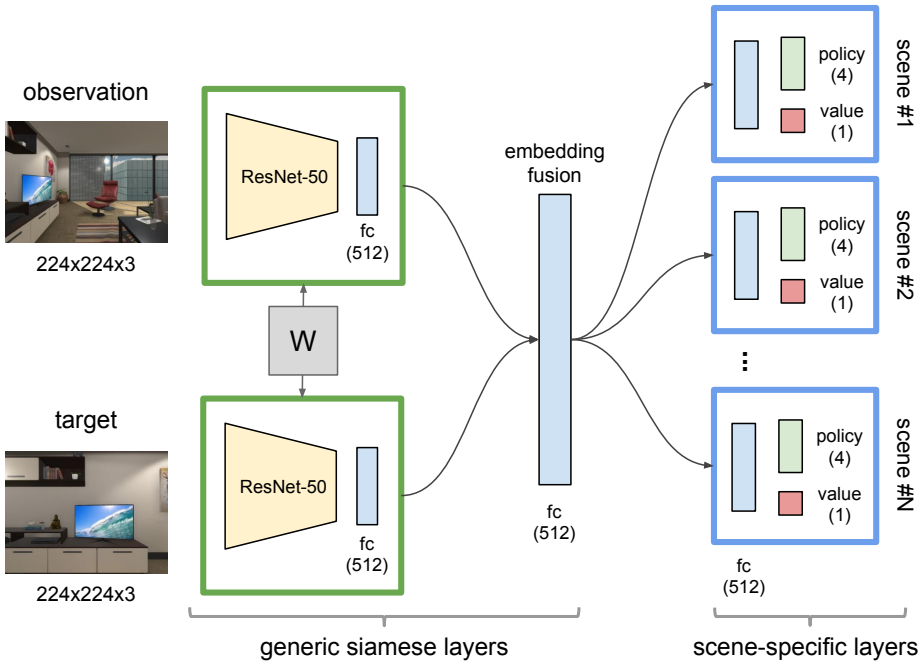


Figure 4.4: Network architecture of our deep siamese actor-critic model. The numbers in parentheses show the output dimensions. Layer parameters in the green squares are shared. The ResNet-50 layers (yellow) are pre-trained on ImageNet and fixed during training.

the agent's current observation and the target. Our approach to reasoning about the spatial arrangement between the current location and the target is to project them into the same embedding space, where their geometric relations are preserved. Deep siamese networks are a type of two-stream neural network models for discriminative embedding learning [34]. We use two streams of weight-shared siamese layers to transform the current state and the target into the same embedding space. Information from both embeddings is fused to form a joint representation. This joint representation is passed through *scene-specific* layers (refer to Figure 4.4). The intention to have scene-specific layers is to capture the special characteristics (e.g., room layouts and object arrangements) of a scene that are crucial for the navigation tasks. Finally, the model generates policy and value outputs similar to the advantage actor-critic models [160]. In this model, targets across all scenes share the same generic siamese layers, and all targets within a scene share the same scene-specific layer. This makes the model better generalize across targets and across scenes.

4.4.4 Training Protocol

Traditional RL models learn for individual tasks in separation, resulting in the inflexibility with respect to goal changes. As our deep siamese actor-critic network shares parameters across different tasks, it can benefit from learning with multiple goals simultaneously. A3C [160] is a type of reinforcement learning model that learns by running multiple copies of training threads in parallel and updates a shared set of model parameters in an asynchronous manner. It has been shown that these parallel training threads stabilize each other, achieving the state-of-the-art performance in the video-game domain. We use a similar training protocol as A3C. However, rather than running copies of a single game, each thread runs with a different navigation target. Thus, gradients are backpropagated from the actor-critic outputs back to the lower-level layers. The scene-specific layers are updated by gradients from the navigation tasks within the scene, and the generic siamese layers are updated by all targets.

4.4.5 Network Architectures

The bottom part of the siamese layers are ImageNet-pretrained ResNet-50 [91] layers (truncated the softmax layer) that produce 2048-d features on a $224 \times 224 \times 3$ RGB image. We freeze these ResNet parameters during training. We concatenate features of 4 history frames to account for the agent's past motions. These 8192-d output vectors from both streams are projected into the 512-d embedding space. The fusion layer takes a 1024-d concatenated embedding of the state and the target, generating a 512-d joint representation. This vector is further passed through two fully-connected scene-specific layers, producing 4 policy outputs (i.e., probability over actions) and a single value output. We train this network with a shared RMSProp optimizer of learning rate 7×10^{-4}.

4.5 Experiments

Our main objective for target-driven navigation is to find the shortest trajectories from the current location to the target. We first evaluate our model with baseline navigation models that are based on heuristics and standard deep RL models. One major advantage of our proposed model is the ability to generalize to new scenes and new targets. We conduct two additional experiments to evaluate the ability of our model to transfer knowledge across targets and across scenes. Also, we show an extension of our model to continuous space. Lastly, we demonstrate the performance of our model

in a complex real setting using a real robot.

4.5.1 Navigation Results

We implement our models in Tensorflow [1] and train them on an Nvidia GeForce GTX Titan X GPU. We follow the training protocol described in Section 4.4.4 to train our deep siamese actor-critic model (see Figure 4.4) with 100 threads, each thread learns for a different target. It takes around 1.25 hours to pass through one million training frames across all threads. We report the performance as the average number of steps (i.e., average trajectory length) it takes to reach a target from a random starting point. The navigation performance is reported on 100 different goals randomly sampled from 20 indoor scenes in our dataset. We compare our final model with heuristic strategies, standard deep RL models, and variations of our model. The models we compare are:

1. **Random walk** is the simplest heuristic for navigation. In this baseline model, the agent randomly draws one out of four actions at each step.

2. **Shortest Path** provides an upper-bound performance for our navigation model. As we discretize the walking space by a constant step length (see Section 4.4.2), we can compute the shortest paths from the starting locations to the target locations. Note that for computing the shortest path, we have access to the full map of the environment, while the input to our system is just an RGB image.

3. **A3C** [160] is an asynchronous advantage actor-critic model that achieves the state-of-the-art results in Atari games. Empirical results show that using more threads improves the data efficiency during training. We thus evaluate A3C model in two setups, where we use 1 thread and 4 threads to train for each target.

4. **One-step Q** [160] is an asynchronous variant of deep Q-network [161].

5. **Target-driven single branch** is a variation of our deep siamese model that does not have scene-specific branches. In this case, all targets will use and update the same scene-specific parameters, including two FC layers and the policy/value output layers.

6. **Target-driven final** is our deep siamese actor-critic model introduced in Section 6.4.

For all learning models, we report their performances after being trained with 100M frames (across all threads). The performance is measured by the average trajectory length (i.e., number

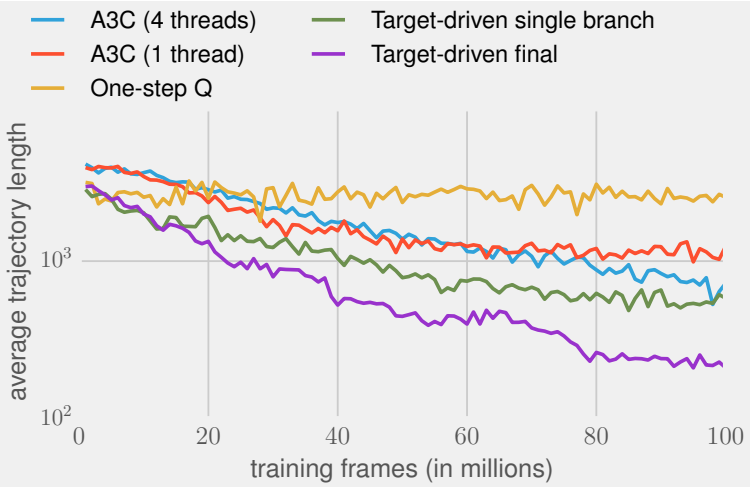


Figure 4.5: Data efficiency of training. Our model learns better navigation policies compared to the state-of-the-art A3C methods [160] after 100M training frames.

of steps taken) over all targets. An episode ends when either the agent reaches the target, or after it takes 10,000 steps. For each target, we randomly initialize the agent's starting locations, and evaluate 10 episodes. The results are listed in Table 4.1.

Table 4.1: Performance of Target-driven Methods and Baselines

Type	Method	Avg. Trajectory Length
Heuristic	Random walk	2744.3
	Shortest path	17.6
Purpose-built RL	One-step Q	2539.2
	A3C (1 thread)	1241.3
	A3C (4 threads)	723.5
Target-driven RL (Ours)	Single branch	581.6
	Final	**210.7**

We analyze the data efficiency of several deep RL models with the learning curves in Figure 4.5. Q-learning suffers from slow convergence. A3C performs better than Q-learning; plus, increasing the number of actor-learning threads per target from 1 to 4 improves learning efficiency. Our proposed target-driven navigation model significantly outperforms standard deep RL models with 100M frames for training. We hypothesize that this is because both the weight sharing scheme across targets and the asynchronous training protocol facilitate learning generalizable knowledge.

In contrast, purpose-built RL models are less data-efficient, as there is no straightforward mechanism to share information across different scenes or targets. The average trajectory length of the final model is three times shorter than the one of the *single branch* model. It justifies the use of scene-specific layers, as it captures particular characteristics of a scene that may vary across scene instances.

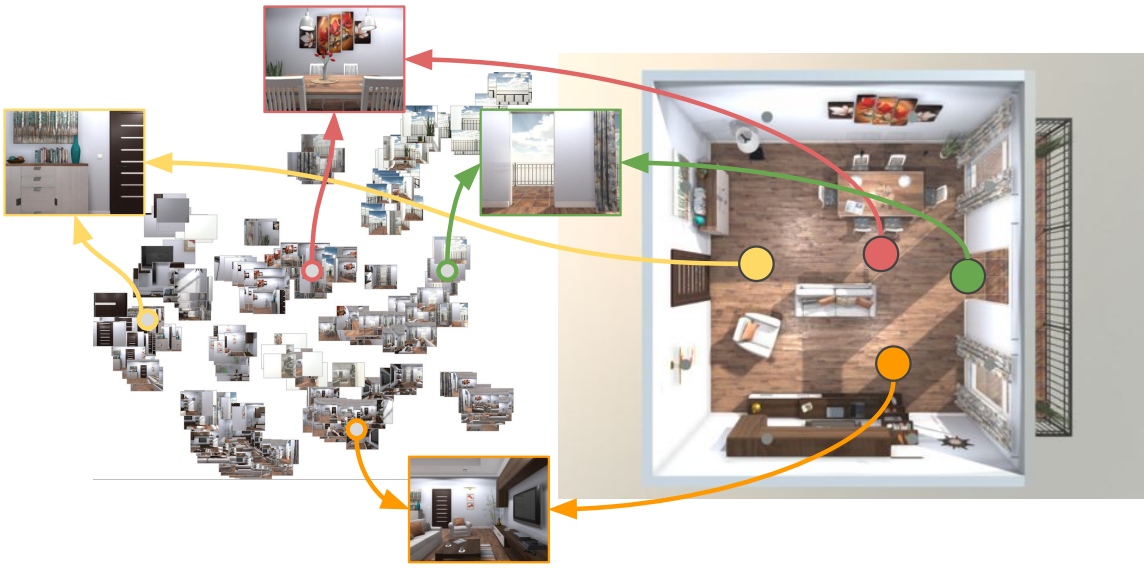


Figure 4.6: t-SNE embeddings of observations in a living room scene. We highlight four observation examples in the projected 2D space and their corresponding locations in the scene (bird's-eye view on the right). This figure shows that our model has learned observation embeddings while preserving their relative spatial layout.

To understand what the model learns, we examine the embeddings learned by generic siamese layers. Figure 4.6 shows t-SNE visualization [241] of embedding vectors computed from observations at different locations at four different orientations. We observe notable spatial correspondence between the spatial arrangement of these embedding vectors and their corresponding t-SNE projections. We therefore hypothesize that the model learns to project observation images into the embedding space while preserving their spatial configuration. To validate this hypothesis, we compare the distance of pairwise projected embeddings and the distance of their corresponding scene coordinates. The Pearson correlation coefficient is 0.62 with p-value less than 0.001, indicating that the embedding space preserves information of the original locations of observations. This means that the model learns a rough map of the environment and has the capability of localization with

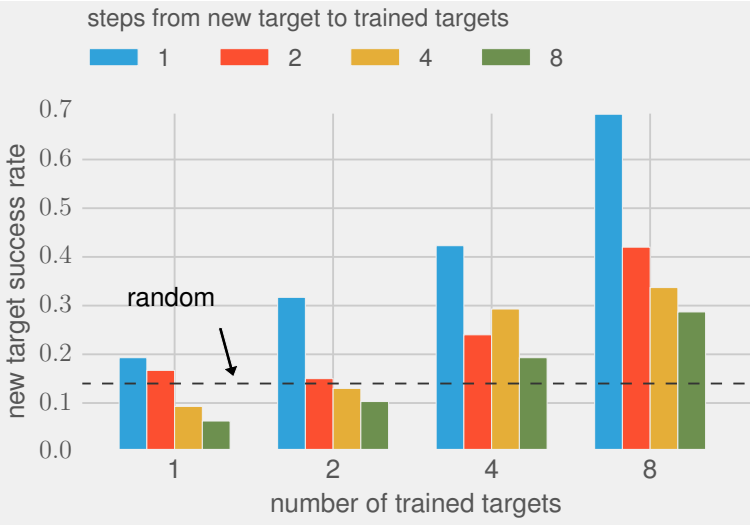


Figure 4.7: Target generalization. Each histogram group reports the success rate of navigation to new targets with certain number of trained targets. The four bars in each group indicate the impact of adjacency between the trained and new targets on generalization performance.

respect to this map.

4.5.2 Generalization Across Targets

In addition to the data efficiency of our target-driven RL models, it has the built-in ability to generalize, which is a significant advantage over the purpose-built baseline models. We evaluate its generalization ability in two aspects: 1. generalizing to new targets within a scene, and 2. generalizing to new scenes. We focus on generalization across targets in this section, and explain scene generalization in the next section.

We test the model to navigate to new targets. These targets are not trained, but they might share common routes with trained targets, thus allowing knowledge to transfer. We take 10 of the largest scenes in our dataset, each having around 15 targets. We gradually increase the number of trained targets (from 1, 2, 4 to 8) using our target-driven model. All models are trained with 20M frames. During testing, we run 100 episodes for each of 10 new targets. These new targets are randomly chosen from a set of locations that have a constant distance (1, 2, 4 and 8 steps) from the nearest trained targets. The results are illustrated in Figure 4.7. We use *success rate* (percentage of trajectories shorter than 500 steps) to measure the performance. We choose this metric due to the bipolar behavior of our model on new targets – it either reaches the new targets quickly, or fails

completely. Thus, this metric is more effective than average trajectory lengths. In Figure 4.7, we observe a consistent trend of increasing success rate, as we increase the number of trained targets (x-axis). Inside each histogram group, the success rate positively correlates with adjacency between trained and new targets. It indicates that the model has a clearer understanding of nearby regions around the trained targets than distant locations.

4.5.3 Generalization Across Scenes

We further evaluate our model's ability to generalize across scenes. As the generic siamese layers are shared over all scenes, we examine the possibility of transferring knowledge from these layers to new scenes. Furthermore, we study how the number of trained scenes would influence the transferability of generic layer parameters. We gradually increase the number of trained scenes from 1 to 16, and test on 4 unseen scenes. We select 5 random targets from each scene for training and testing. To adapt to unseen scenes, we train the scene-specific layers while fixing generic siamese layers. Figure 4.8 shows the results. We observe faster convergence as the number of trained scenes grows. Compared to training from scratch, transferring generic layers significantly improves data efficiency for learning in new environments. We also evaluate the *single branch* model in the same setup. As the *single branch* model includes a single scene-specific layer, we can apply a trained model (trained on 16 scenes) to new scenes without extra training. However, it results in worse performance than chance, indicating the importance of adapting scene-specific layers. The *single branch* model leads to slightly faster convergence than training from scratch, yet far slower than our final model.

4.5.4 Continuous Space

The space discretization eliminates the need for handling complex system dynamics, such as noise in motor control. In this section, we show empirical results that the same model is capable of coping with more challenging continuous space.

To illustrate this, we train the same target-driven model for a door-finding task in a large living room scene, where the goal is to arrive at the balcony through a door. We use the same 4 actions as before (see Section 4.4.2); however, the agent's moves and turns are controlled by the physics engine. In this case, the method should explicitly handle forces and collisions, as the agent may be stopped by obstacles or slide along heavy objects. Although this setting requires significantly more training frames (around 50M) to train for a single target, the same model learns to reach the

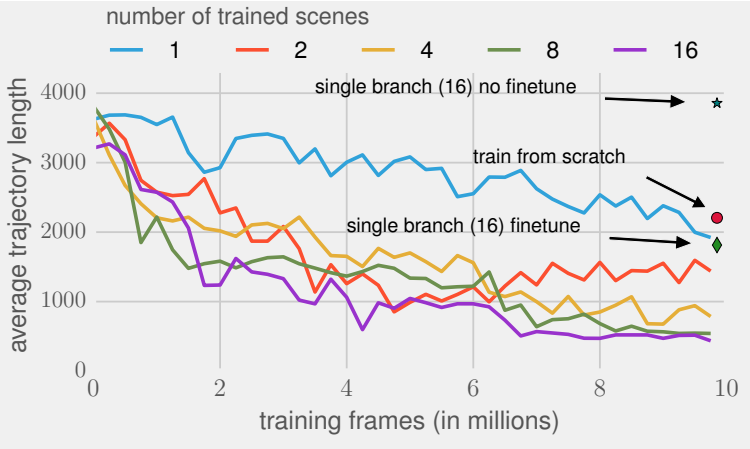


Figure 4.8: Scene generalization. We compare the data efficiency for adapting trained navigation models to unseen scenes. As the number of trained scene instances increases, fine-tuning the scene-specific layers becomes faster.

door in average 15 steps, whereas random agents take 719 steps on average. We provide sample test episodes in the video: https://youtu.be/SmBxMDiOrvs.

4.5.5 Robot Experiment

To validate the generalization of our method to real world settings, we perform an experiment by using a SCITOS mobile robot modified by [35] (see Figure 4.9). We train our model in three different settings: 1) training on real images from scratch; 2) training only scene-specific layers while freezing generic layer parameters trained on 20 simulated scenes; and 3) training scene-specific layers and fine-tuning generic layer parameters.

We train our model (with backward action disabled) on 28 discrete locations in the scene, which are roughly 30 inches apart from each other in each dimension. At each location, the robot takes 4 RGB images (90 degrees apart) using its head camera. During testing, the robot moves and turns based on the model's predictions. We evaluate the robot with two targets in the room: *door* and *microwave*. Although the model is trained on a discretized space, it exhibits robustness towards random starting points, noisy dynamics, varying step lengths, changes in illumination and object layouts, etc. Example test episodes are provided in the video. All three setups converge to nearly-optimal policy due to the small scale of the real scene. However, we find that transferring and fine-tuning parameters from simulation to real data offers the fastest convergence out of these three

Figure 4.9: Robot experiment setup. Our experiments are conducted on a SCITOS mobile robot. On the left, we show a picture of the SCITOS robot. On the right, we show the test environment and one target (microwave) that we have used for evaluation.

setups (44% faster than training from scratch). This provides supportive evidence on the value of simulations in learning real-world interactions and shows the possibility of generalization from simulation to real images using a small amount of fine-tuning.

4.6 Discussions

In this chapter, we proposed a deep reinforcement learning (DRL) framework for target-driven visual navigation. The state-of-the-art DRL methods are typically applied to video games and environments that do not mimic the distribution of natural images. This work is a step towards more realistic settings.

The state-of-the-art DRL methods have some limitations that prevent them from being applied to realistic settings. In this work we have addressed some of these limitations. We addressed generalization across targets and across scenes, improved data efficiency compared to the state-of-the-art DRL methods, and provided the AI2-THOR framework that enables inexpensive and efficient collection of action and interaction data.

Our experiments showed that our method generalizes to new targets and scenes that are not used during the end-to-end training of the model. We also showed our method converges with much fewer training samples compared to the state-of-the-art DRL methods. Furthermore, we showed that the method works in both discrete and continuous domains. We also showed that a model that

is trained on simulation can be adapted to a real robot with a small amount of fine-tuning. We provided visualizations that show that our DRL method implicitly performs localization and mapping. Finally, our method is end-to-end trainable. Unlike the common visual navigation methods, it does not require explicit feature matching or 3D reconstruction of the environment.

Chapter 5

Visual Semantic Planning

5.1 Introduction

Humans demonstrate levels of visual understanding that go well beyond current formulations of mainstream vision tasks (e.g., object detection, scene recognition, image segmentation). A key element to visual intelligence is the ability to interact with the environment and *plan* a sequence of actions to achieve specific goals; This, in fact, is central to the survival of agents in dynamic environments [6, 171].

Visual semantic planning, the task of interacting with a visual world and predicting a sequence of actions that achieves a desired goal, involves addressing several challenging problems. For example, imagine the simple task of `putting the bowl in the microwave` in the visual dynamic environment depicted in Figure 5.1. A successful plan involves first finding the bowl, navigating to it, then grabbing it, followed by finding and navigating to the microwave, opening the microwave, and finally putting the bowl in the microwave.

The *first* challenge in visual planning is that performing each of the above actions in a visual dynamic environment requires deep visual understanding of that environment, including the set of possible actions, their preconditions and effects, and object affordances. For example, to *open a microwave* an agent needs to know that it should be in front of the microwave, and it should be aware of the state of the microwave and not try to open an already opened microwave. Long explorations that are required for some tasks imposes the *second* challenge. The variability of visual observations and possible actions makes naïve exploration intractable. To *find a cup*, the agent might need to search several cabinets one by one. The *third* challenge is emitting a sequence of actions such that the agent ends in the goal state and the effects of the preceding actions meet

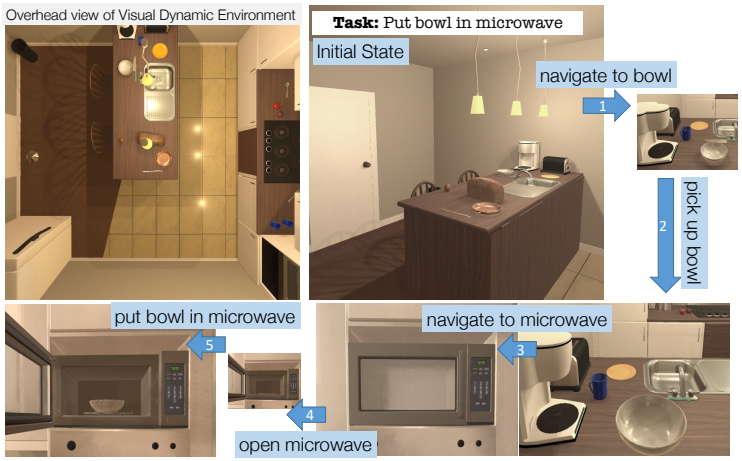


Figure 5.1: Given a task and an initial configuration of a scene, our agent learns to interact with the scene and predict a sequence of actions to achieve the goal based on visual inputs.

the preconditions of the proceeding ones. Finally, a satisfactory solution to visual planning should enable cross task transfer; previous knowledge about one task should make it easier to learn the next one. This is the *fourth* challenge.

In this chapter, we address visual planning as a policy learning problem. We mainly focus on high-level actions and do not take into account the low-level details of motor control and motion planning. Visual Semantic Planning (VSP) is the task of predicting a sequence of semantic actions that moves an agent from a random initial state in a visual dynamic environment to a given goal state.

To address the first challenge, one needs to find a way to represent the required knowledge of objects, actions, and the visual environment. One possible way is to learn these from still images or videos [58, 240, 251]. But we argue that learning high-level knowledge about actions and their preconditions and effects requires an active and prolonged interaction with the environment. We take an interaction-centric approach where we learn this knowledge through interacting with the *visual dynamic environment*. Learning by interaction on real robots has limited scalability due to the complexity and cost of robotics systems [187, 188, 231]. A common treatment is to use simulation as *mental rehearsal* before real-world deployment [17, 112, 138, 259, 271]. For this purpose, we use the THOR framework [271], extending it to enable interactions with objects, where an action is specified as its pre- and post-conditions in a formal language.

To address the second and third challenges, we cast VSP as a policy learning problem, typically

tackled by reinforcement learning [51, 80, 116, 146, 161, 219]. To deal with the large action space and delayed rewards, we use imitation learning to bootstrap reinforcement learning and to guide exploration. To address the fourth challenge of cross task generalization [133], we develop a deep predictive model based on successor representations [41, 130] that decouple environment dynamics and task rewards, such that knowledge from trained tasks can be transferred to new tasks with theoretical guarantees [13].

In summary, we address the problem of visual semantic planning and propose an interaction-centric solution. Our proposed model obtains near optimal results across a spectrum of tasks in the challenging THOR environment. Our results also show that our deep successor representation offers crucial transferability properties. Finally, our qualitative results show that our learned representation can encode visual knowledge of objects, actions, and environments.

5.2 Related Work

Task planning. Task-level planning [50, 62, 108, 226, 227] addresses the problem of finding a high-level plan for performing a task. These methods typically work with high-level formal languages and low-dimensional state spaces. In contrast, visual semantic planning is particularly challenging due to the high dimensionality and partial observability of visual input. In addition, our method facilitates generalization across tasks, while previous methods are typically designed for a specific environment and task.

Perception and interaction. Our work integrates perception and interaction, where an agent actively interfaces with the environment to learn policies that map pixels to actions. The synergy between perception and interaction has drawn an increasing interest in the vision and robotics community. Recent work has enabled faster learning and produced more robust visual representations [3, 153, 187] through interaction. Some early successes have been shown in physical understanding [47, 138, 143, 163] by interacting with an environment.

Deep reinforcement learning. Recent work in reinforcement learning has started to exploit the power of deep neural networks. Deep RL methods have shown success in several domains such as video games [161], board games [219], and continuous control [146]. Model-free RL methods (e.g., [146, 160, 161]) aim at learning to behave solely from actions and their environment feedback; while model-based RL approaches (e.g., [46, 212, 233]) also estimate a environment model. Successor representation (SR), proposed by Dayan [41], can be considered as a hybrid approach of model-based and model-free RL. Barreto *et al.* [13] derived a bound on value functions of an optimal

Figure 5.2: Example images that demonstrate the state changes before and after an object interaction from each of the six action types in our framework. Each action changes the visual state and certain actions may enable further interactions such as opening the fridge before taking an object from it.

policy when transferring policies using successor representations. Kulkarni *et al.* [130] proposed a method to learn deep successor features. In this work, we propose a new SR architecture with significantly reduced parameters, especially in large action spaces, to facilitate model convergence. We propose to first train the model with imitation learning and fine-tune with RL. It enables us to perform more realistic tasks and offers significant benefits for transfer learning to new tasks.

Learning from demonstrations. Expert demonstrations offer a source of supervision in tasks which must usually be learned with copious random exploration. A line of work interleaves policy execution and learning from expert demonstration that has achieved good practical results [39, 201]. Recent works have employed a series of new techniques for imitation learning, such as generative adversarial networks [95, 144], Monte Carlo tree search [82] and guided policy search [140], which learn end-to-end policies from pixels to actions.

Synthetic data for visual tasks. Computer games and simulated platforms have been used for training perceptual tasks, such as semantic segmentation [87], pedestrian detection [156], pose estimation [177], and urban driving [33, 197, 200, 218]. In robotics, there is a long history of using simulated environments for learning and testing before real-world deployment [119]. Several interactive platforms have been proposed for learning control with visual inputs [17, 112, 138, 259, 271]. Among these, THOR [271] provides high-quality realistic indoor scenes. Our work extends THOR with a new set of actions and the integration of a planner.

5.3 Extending the AI2-THOR Framework

To enable interactions with objects and with the environment, we extend the AI2-THOR framework [271], which has been used for learning visual navigation tasks. Our extension includes new object states, and a discrete description of the scene in a planning language [62].

5.3.1 Scenes

In this work, we focus on *kitchen* scenes, as they allow for a variety of tasks with objects from many categories. Our extended THOR framework consists of 10 individual kitchen scenes. Each scene contains an average of 53 distinct objects with which the agent can interact. The scenes are developed using the Unity 3D game engine.

5.3.2 Objects and Actions

We categorize the objects by their affordances [75], i.e., the plausible set of actions that can be performed. For the tasks of interest, we focus on two types of objects: 1) *items* that are small objects (mug, apple, etc.) which can be picked up, held, and moved by the agent to various locations in the scene, and 2) *receptacles* that are large objects (table, sink, etc.) which are stationary and can hold a fixed capacity of *items*. A subset of *receptacles*, such as fridges and cabinets, are *containers*. These *containers* have doors that can be opened and closed. The agent can only put an *item* in a *container* when it is open. We assume that the agent can hold at most one *item* at any point. We further define the following actions to interact with the objects:

1. Navigate {*receptacle*}: moving from the current location of the agent to a location near the *receptacle*;
2. Open {*container*}: opening the door of a *container* in front of an agent;
3. Close {*container*}: closing the door of a *container* in front of an agent;
4. Pick Up {*item*}: picking up an *item* in field of view;
5. Put {*receptacle*}: putting a held item in the *receptacle*;
6. Look Up and Look Down: tilting the agent's gaze 30 degrees up or down.

In summary, we have six action types, each taking a corresponding type of action arguments. The combination of actions and arguments results in a large action set of 80 per scene on average. Figure 5.2 illustrates example scenes and the six types of actions in our framework. Our definition of action space makes two important abstractions to make learning tractable: 1) it abstracts away

from navigation, which can be tackled by a subroutine using existing methods such as [271]; and 2) it allows the model to learn with semantic actions, abstracting away from continuous motions, e.g., the physical movement of a robot arm to grasp an object. A common treatment for this abstraction is to "fill in the gaps" between semantic actions with callouts to a continuous motion planner [108, 226]. It is evident that not all actions can be performed in every situation. For example, the agent cannot pick up an *item* when it is out of sight, or put a tomato into fridge when the fridge door is closed. To address these requirements, we specify the pre-conditions and effects of actions. Next we introduce an approach to declaring them as logical rules in a planning language. These rules are only encoded in the environment but not exposed to the agent. Hence, the agent must learn them through interaction.

5.3.3 Planning Language

The problem of generating a sequence of actions that leads to the goal state has been formally studied in the field of automated planning [74]. Planning languages offer a standard way of expressing an automated planning problem instance, which can be solved by an off-the-shelf planner. We use STRIPS [62] as the planning language to describe our visual planning problem.

In STRIPS, a planning problem is composed of a description of an initial state, a specification of the goal state(s), and a set of actions. In visual planning, the initial state corresponds to the initial configuration of the scene. The specification of the goal state is a boolean function that returns true on states where the task is completed. Each action is defined by its precondition (conditions that must be satisfied before the action is performed) and postcondition (changes caused by the action). The STRIPS formulation enables us to define the rules of the scene, such as object affordances and causality of actions.

5.4 Model

We first outline the basics of policy learning in Section 5.4.1. Next we formulate the *visual semantic planning* problem as a policy learning problem and describe our model based on successor representation. Later we propose two protocols of training this model using imitation learning (IL) and reinforcement learning (RL). To this end, we use IL to bootstrap our model and use RL to further improve its performance.

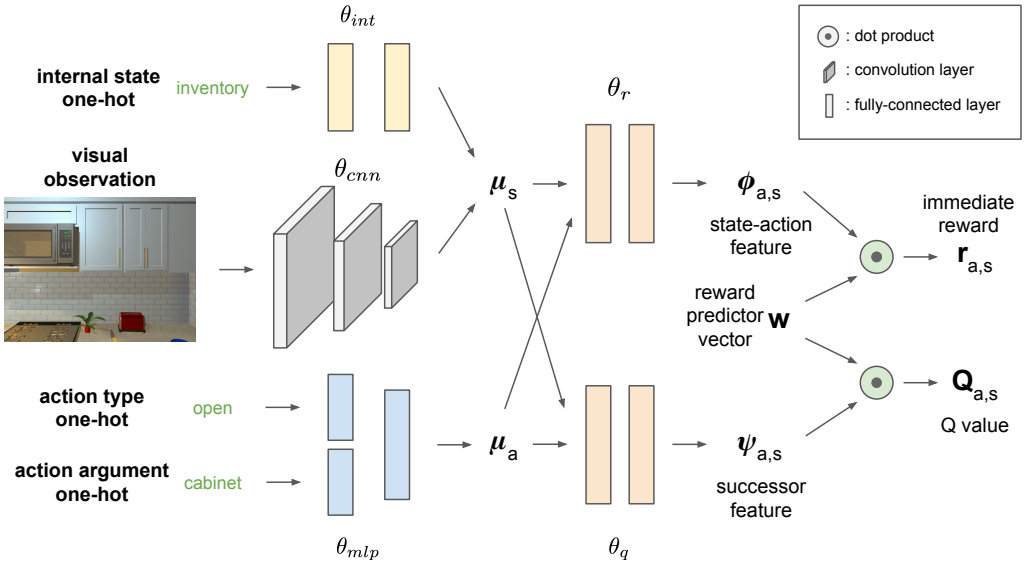


Figure 5.3: Overview of the network architecture of our successor representation (SR) model. Our network takes in the current state as well as a specific action and predicts an immediate reward $r_{a,s}$ as well as a discounted future reward $Q_{a,s}$, performing this evaluation for each action. The learned policy π takes the argmax over all Q values as its chosen action.

5.4.1 Successor Representation

We formulate the agent's interactions with an environment as a *Markov Decision Process* (MDP), which can be specified by a tuple $(\mathscr{S}, \mathscr{A}, p, r, \gamma)$. $\mathscr{S}$ and $\mathscr{A}$ are the sets of states and actions. For $s \in \mathscr{S}$ and $a \in \mathscr{A}$, $p(s'|s,a)$ defines the probability of transiting from the state s to the next state $s' \in \mathscr{S}$ by taking action a. $r(s,a)$ is a real-value function that defines the expected immediate reward of taking action a in state s. For a state-action trajectory, we define the future discounted *return* $R = \sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)$, where $\gamma \in [0,1]$ is the discount factor, which trades off the importance of immediate rewards versus future rewards.

A *policy* $\pi : \mathscr{S} \to \mathscr{A}$ defines a mapping from states to actions. The goal of policy learning is to find the optimal policy π^* that maximizes the future discounted return R starting from state s_0 and following the policy π^*. Instead of directly optimizing a parameterized policy, we take a value-based approach. We define a state-action value function $Q^\pi : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ under a policy π:

$$Q^\pi(s,a) = \mathbb{E}^\pi[R|s_0 = s, a_0 = a], \tag{5.1}$$

i.e., the expected episode return starting from state s, taking action a, and following policy π. The Q value of the optimal policy π^* obeys the Bellman equation [231]:

$$Q^{\pi^*}(s,a) = \mathbb{E}^{\pi^*}[r(s,a) + \gamma \max_{a'} Q(s',a')] \tag{5.2}$$

In deep Q networks [161], Q functions are approximated by a neural network $Q(s,a|\theta)$, and can be trained by minimizing the ℓ_2-distance between both sides of the Bellman equation in Equation (5.2). Once we learn Q^{π^*}, the optimal action at state s can be selected by $a^* = \arg\max_a Q^{\pi^*}(s,a)$.

Successor representation (SR), proposed by Dayan [41], uses a similar value-based formulation for policy learning. It differs from traditional Q learning by factoring the value function into a dot product of two components: a reward predictor vector $\mathbf{w}$ and a predictive successor feature $\psi(s,a)$. To derive the SR formulation, we start by factoring the immediate rewards such that

$$r(s,a) = \phi(s,a)^T \mathbf{w}, \tag{5.3}$$

where $\phi(s,a)$ is a *state-action feature*. We expand Equation (5.1) using this reward factorization:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}^{\pi}[\sum_{i=0}^{\infty} \gamma^i r(s_i,a_i)|s_0 = s, a_0 = a] \\ &= \mathbb{E}^{\pi}[\sum_{i=0}^{\infty} \gamma^i \phi(s_i,a_i)^T \mathbf{w}|s_0 = s, a_0 = a] \\ &= \mathbb{E}^{\pi}[\sum_{i=0}^{\infty} \gamma^i \phi(s_i,a_i)|s_0 = s, a_0 = a]^T \mathbf{w} \\ &= \psi^{\pi}(s,a)^T \mathbf{w} \end{aligned} \tag{5.4}$$

We refer to $\psi(s,a)^{\pi} = \mathbb{E}^{\pi}[\sum_{i=0}^{\infty} \gamma^i \phi_{s,a}|s_0 = s, a_0 = a]$ as the *successor features* of the pair (s,a) under policy π.

Intuitively, the successor feature $\psi^{\pi}(s,a)$ summarizes the environment dynamics under a policy π in a state-action feature space, which can be interpreted as the expected future "feature occupancy". The reward predictor vector $\mathbf{w}$ induces the structure of the reward functions, which can be considered as an embedding of a task. Such decompositions have been shown to offer several advantages, such as being adaptive to changes in distal rewards and apt to option discovery [130]. A theoretical result derived by Barreto *et al.*implies a bound on performance guarantee when the agent transfers a policy from a task t to a similar task t', where task similarity is determined by the ℓ_2-distance of the corresponding $\mathbf{w}$ vectors between these two tasks t and t' [13]. Successor

representation thus provides a generic framework for policy transfer in reinforcement learning.

5.4.2 Deep Successor Model

We formulate the problem of *visual semantic planning* as a policy learning problem. Formally, we denote a task by a Boolean function $t : \mathscr{S} \to \{0,1\}$, where a state s completes the task t iff $t(s) = 1$. The goal is to find an optimal policy π^*, such that given an initial state s_0, π^* generates a state-action trajectory $\mathscr{T} = \{(s_i, a_i) \,|\, i = 0 \ldots T\}$ that maximizes the sum of immediate rewards $\sum_{i=0}^{T-1} r(s_i, a_i)$, where $t(s_{0 \ldots T-1}) = 0$ and $t(s_T) = 1$.

We parameterize such a policy using the successor representation (SR) model from the previous section. We develop a new neural network architecture to learn ϕ, ψ and $\mathbf{w}$. The network architecture is illustrated in Figure 5.3. In THOR, the agent's observations come from a first-person RGB camera. We also pass the agent's internal state as input, expressed by one-hot encodings of the held object in its inventory. The action space is described in Section 5.3.2. We start by computing embedding vectors for the states and the actions. The image is passed through a 3-layer convolutional encoder, and the internal state through a 2-layer MLP, producing a state embedding $\mu_s = f(s; \theta_{cnn}, \theta_{int})$. The action $a = [a_{type}, a_{arg}]$ is encoded as one-hot vectors and passed through a 2-layer MLP encoder that produces an action embedding $\mu_a = g(a_{type}, a_{arg}; \theta_{mlp})$. We fuse the state and action embeddings and generate the state-action feature $\phi_{s,a} = h(\mu_s, \mu_a; \theta_r)$ and the successor feature $\psi_{s,a} = m(\mu_s, \mu_a; \theta_q)$ in two branches. The network predicts the immediate reward $r_{s,a} = \phi_{s,a}^T \mathbf{w}$ and the Q value under the current policy $Q_{s,a} = \psi_{s,a}^T \mathbf{w}$ using the decomposition from Equation (5.3) and (5.4).

5.4.3 Imitation Learning

Our SR-based policy can be learned in two fashions. First, it can be trained by imitation learning (IL) under the supervision of the trajectories of an optimal planner. Second, it can be learned by trial and error using reinforcement learning (RL). In practice, we find that the large action space in THOR makes RL from scratch intractable due to the challenge of exploration. The best model performance is produced by IL bootstrapping followed by RL fine-tuning. Given a task, we generate a state-action trajectory:

$$\mathscr{T} = \{(s_0, a_0), \{(s_1, a_1), \ldots, (s_{T-1}, a_{T-1}), (s_T, \emptyset)\} \tag{5.5}$$

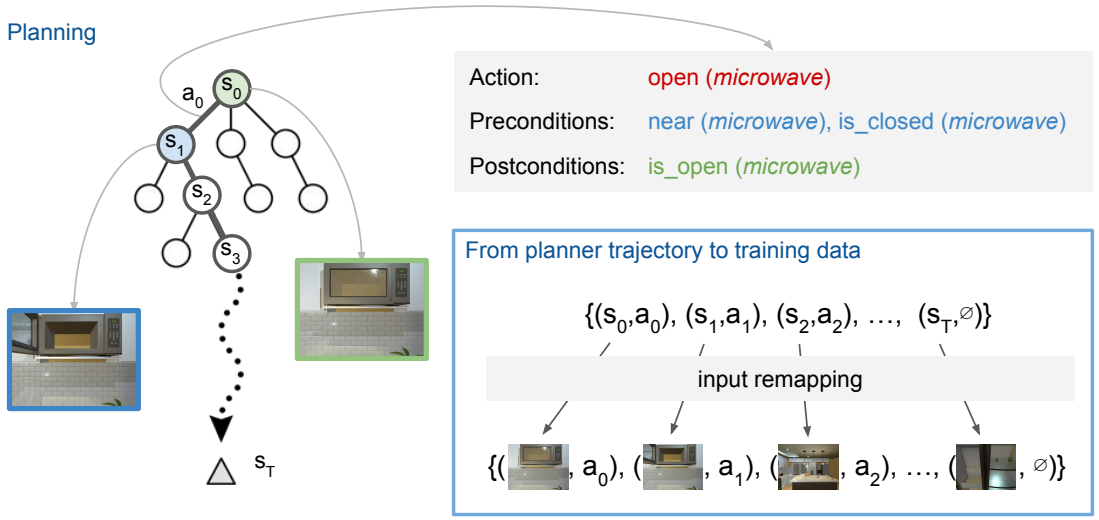


Figure 5.4: We use a planner to generate a trajectory from an initial state-action pair (s_0, a_0) to a goal state s_T. We describe each scene in a STRIPS-based planning language, where actions are specified by their pre- and post-conditions (see Section 5.3.3). We perform input remapping, illustrated in the blue box, to obtain the image-action pairs from the trajectory as training data. After performing an action, we update the plan and repeat.

using the planner from the initial state-action pair (s_0, a_0) to the goal state s_T (no action is performed at terminal states). This trajectory is generated on a low-dimensional state representation in the STRIPS planner (Section 5.3.3). Each low-dimensional state corresponds to an RGB image, i.e., the agent's visual observation. During training, we perform *input remapping* to supervise the model with image-action pairs rather than feeding the low-dimensional planner states to the network. To fully explore the state space, we take planner actions as well as random actions off the optimal plan. After each action, we recompute the trajectory. This process of generating training data from a planner is illustrated in Figure 5.4. Each state-action pair is associated with a true immediate reward $\hat{r}_{s,a}$. We use the mean squared loss function to minimize the error of reward prediction:

$$\mathscr{L}_r = \frac{1}{T}\sum_{i=0}^{T-1}(\hat{r}_{s,a} - \phi_{s,a}^T \mathbf{w})^2. \tag{5.6}$$

Following the REINFORCE rule [231], we use the discounted return along the trajectory $\mathscr{T}$ as an unbiased estimate of the true Q value: $\hat{Q}_{s,a} \approx \sum_{i=0}^{T-1} \gamma^i \hat{r}_{s,a}$. We use the mean squared loss to minimize the error of Q prediction:

$$\mathscr{L}_Q = (\hat{Q}_{s,a} - \psi_{s,a}^T \mathbf{w})^2 \tag{5.7}$$

The final loss on the planner trajectory $\mathscr{T}$ is the sum of the reward loss and the Q loss: $\mathscr{L}_{\mathscr{T}} = \mathscr{L}_r + \mathscr{L}_Q$. Using this loss signal, we train the whole SR network on a large collection of planner trajectories starting from random initial states.

5.4.4 Reinforcement Learning

When training our SR model using RL, we can still use the mean squared loss in Equation (5.6) to supervise the learning of reward prediction branch for ϕ and $\mathbf{w}$. However, in absence of expert trajectories, we would need an iterative way to learn the successor features ψ. Rewriting the Bellman equation in Equation (5.2) with the SR factorization, we can obtain an equality on ϕ and ψ:

$$\psi^{\pi^*}(s,a) = \mathbb{E}^{\pi^*}[\phi(s,a) + \gamma\psi(s',a')] \tag{5.8}$$

where $a' = \arg\max_a \psi(s',a)^T \mathbf{w}$. Similar to DQN [161], we minimize the ℓ_2-loss between both sides of Equation (5.8):

$$L_{SR} = \mathbb{E}^{\pi}[(\phi_{s,a} + \gamma\psi_{s',a'} - \psi^{\pi}_{s,a})^2] \tag{5.9}$$

We use a similar procedure to Kulkarni *et al.* [130] to train our SR model. The model alternates training between the reward branch and the SR branch. At each iteration, a mini-batch is randomly drawn from a replay buffer of past experiences [161] to perform one SGD update.

5.4.5 Transfer with Successor Features

A major advantage of successor features is its ability to transfer across tasks by exploiting the structure shared by the tasks. Given a fixed state-action representation ϕ, let M_ϕ be the set of all possible MDPs induced by ϕ and all instantiations of the reward prediction vectors $\mathbf{w}$. Assume that π_i^* is the optimal policy of the i-th task in the set $\{M_i \in M_\phi | i = 1,\dots n\}$. Let M_{n+1} to be a new task. We denote $Q_{n+1}^{\pi_i^*}$ as the value function of executing the optimal policy of the task M_i on the new task M_{n+1}, and $\tilde{Q}_{n+1}^{\pi_i^*}$ as an approximation of $Q_{n+1}^{\pi_i^*}$ by our SR model. Given a bound on the approximations such that

$$|Q_{n+1}^{\pi_i^*}(s,a) - \tilde{Q}_{n+1}^{\pi_i^*}(s,a)| \leq \varepsilon \quad \forall s \in \mathscr{S}, a \in \mathscr{A}, i = 1,\dots,n,$$

we define a policy π' for the new task M_{n+1} using $\tilde{Q}_{1,\dots,n}$, where $\pi'(s) = \arg\max_a \max_i \tilde{Q}_{n+1}^{\pi_i^*}(s,a)$. Theorem 2 in Barreto *et al.* [13] implies a bound of the gap between value functions of the optimal

policy π^*_{n+1} and the policy π':

$$Q^{\pi^*_{n+1}}_{n+1}(s,a) - Q^{\pi'}_{n+1}(s,a) \leq \frac{2\phi_m}{1-\gamma}(\min_i ||\mathbf{w}_i - \mathbf{w}_{n+1}|| + \varepsilon),$$

where $\phi_m = \max_{s,a} ||\phi(s,a)||$. This result serves the theoretical foundation of policy transfer in our SR model. In practice, when transferring to a new task while the scene dynamics remain the same, we freeze all model parameters except the single vector $\mathbf{w}$. This way, the policy of the new task can be learned with substantially higher sample efficiency than training a new network from scratch.

5.4.6 Implementation Details

We feed a history of the past four observations, converted to grayscale, to account for the agent's motions. We use a time cost of -0.01 to encourage shorter plans and a task completion reward of 10.0. We train our model with imitation learning for 500k iterations with a batch size of 32, and a learning rate of 1e-4. We also include the successor loss in Equation (5.9) during imitation learning, which helps learn better successor features. We subsequently fine-tune the network with reinforcement learning with 10,000 episodes.

5.5 Experiments

We evaluate our model using the extended THOR framework on a variety of household tasks. We compare our method against standard reinforcement learning techniques as well as with non-successor based deep models. The tasks compare the different methods' abilities to learn across varying time horizons. We also demonstrate the SR network's ability to efficiently adapt to new tasks. Finally, we show that our model can learn a notion of object affordance by interacting with the scene.

5.5.1 Quantitative Evaluation

We examine the effectiveness of our model and baseline methods on a set of tasks that require three levels of planning complexity in terms of optimal plan length.

Experiment Setup We explore the two training protocols introduced in Section 5.4 to train our SR model:

1. **RL**: we train the model solely based on trial and error, and learn the model parameters with RL update rules.

	Easy		Medium		Hard	
	Success Rate	**Mean (σ) Episode Length**	**Success Rate**	**Mean (σ) Episode Length**	**Success Rate**	**Mean (σ) Episode Length**
Random Action	1.00	696.33 (744.71)	0.00	-	0.04	2827.08 (927.84)
Random Valid Action	1.00	64.03 (68.04)	0.02	3897.50 (548.50)	0.36	2194.83 (1401.72)
A3C [160]	0.96	101.12 (151.04)	0.00	-	0.04	2674.29 (4370.40)
CLS-MLP	1.00	**2.42** (0.70)	0.65	256.32 (700.78)	0.65	475.86 (806.42)
CLS-LSTM	1.00	2.86 (0.37)	**0.80**	314.05 (606.25)	**0.66**	136.94 (523.60)
SR IL (ours)	1.00	2.70 (1.06)	**0.80**	32.32 (29.22)	0.65	**34.25** (63.81)
SR IL + RL (ours)	1.00	2.57 (1.04)	**0.80**	**26.56** (3.85)	-	-
Optimal planner	1.00	2.36 (1.04)	1.00	12.10 (6.16)	1.00	14.13 (9.09)

Table 5.1: Results of evaluating the model on the easy, medium, and hard tasks. For each task, we evaluate how many out of the 100 episodes were completed (success rate) and the mean and standard deviation for successful episode lengths. The numbers in parentheses show the standard deviations. We do not fine-tune our SR IL model for the hard task.

2. **IL**: we use the planner to generate optimal trajectories starting from a large collection of random initial state-action pairs. We use the imitation learning methods to train the networks using supervised losses.

Empirically, we find that training with reinforcement learning from scratch cannot handle the large action space. Thus, we report the performance of our SR model trained with imitation learning (SR IL) as well as with additional reinforcement learning fine-tuning (SR IL + RL).

We compare our SR model with the state-of-the-art deep RL model, A3C [160], which is an advantage-based actor-critic method that allows the agent to learn from multiple copies of simulation while updating a single model in an asynchronous fashion. A3C establishes a strong baseline for reinforcement learning. We further use the same architecture to obtain two imitation learning (behavior cloning) baselines. We use the same A3C network structure to train a softmax classifier that predicts the planner actions given an input. The network predicts both the action types (e.g., Put) and the action arguments (e.g., *apple*). We call this baseline CLS-MLP. We also investigate the role of memory in these models. To do this, we add an extra LSTM layer to the network before action outputs, called CLS-LSTM. We also include simple agents that take random actions and take random valid actions at each time step.

Levels of task difficulty We evaluate all of the models with three levels of task difficulty based on the length of the optimal plans and the source of randomization:

1. **Level 1 (Easy)**: Navigate to a *container* and toggle its state. A sample task would be `go to the microwave and open it if it is closed, close it otherwise.` The initial location of the agent and all *container* states are randomized. This task requires identifying object states and reasoning about action preconditions.

2. **Level 2 (Medium)**: Navigate to multiple *receptacles*, collect *items*, and deposit them in a *receptacle*. A sample task here is `pick up three mugs from three cabinets and put them in the sink.` Here we randomize the agent's initial location, while the item locations are fixed. This task requires a long trajectory of correct actions to complete the goal.

3. **Level 3 (Hard)**: Search for an *item* and put it in a *receptacle*. An example task is `find the apple and put it in the fridge.` We randomize the agent's location as well as the location of all items. This task is especially difficult as it requires longer-term memory to account for partial observability, such as which cabinets have previously been checked.

We evaluate all of the models on 10 easy tasks, 8 medium tasks, and 7 hard tasks, each across 100 episodes. Each episode terminates when a goal state is reached. We consider an episode fails if it does not reach any goal state within 5,000 actions. We report the episode success rate and mean episode length as the performance metrics. We exclude these failed episodes in the mean episode length metric. For the easy and medium tasks, we train the imitation learning models to mimic the optimal plans. However for the hard tasks, imitating the optimal plan is infeasible, as the location of the object is uncertain. In this case, the target object is likely to hide in a cabinet or a fridge which the agent cannot see. Therefore, we train the models to imitate a plan which searches for the object from all the *receptacles* in a fixed order. For the same reason, we do not perform RL fine-tuning for the hard tasks.

Table 5.1 summarizes the results of these experiments. Pure RL-based methods struggle with the medium and hard tasks because the action space is so large that naïve exploration rarely, if ever, succeeds. Comparing CLS-MLP and CLS-LSTM, adding memory to the agent helps improving success rate on medium tasks as well as completing tasks with shorter trajectories in hard tasks. Overall, the SR methods outperform the baselines across all three task difficulties. Fine-tuning the SR IL model with reinforcement learning further reduces the number of steps towards the goal.

5.5.2 Task Transfer

One major benefit of the successor representation decomposition is its ability to transfer to new tasks while only retraining the reward prediction vector $\mathbf{w}$, while freezing the successor features. We examine the sample efficiency of adapting a trained SR model on multiple novel tasks in the same scene. We examine policy transfer in the hard tasks, as the scene dynamics of the searching policy retains, even when the objects to be searched vary. We evaluate the speed at which the SR

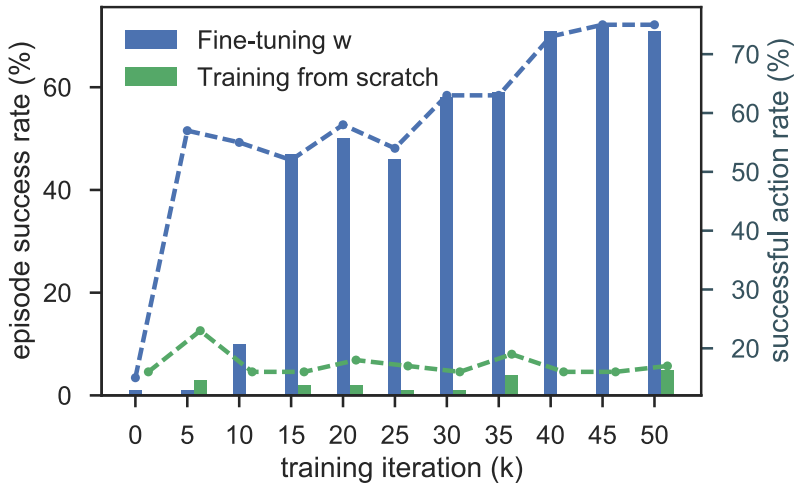


Figure 5.5: We compare updating **w** with retraining the whole network for new hard tasks in the same scene. By using successor features, we can quickly learn an accurate policy for the new item. Bar charts correspond to the episode success rates, and line plots correspond to successful action rate.

model converges on a new task by fine-tuning the **w** vector versus training the model from scratch. We take a policy for searching a bowl in the scene and substituting four new items (lettuce, egg, container, and apple) in each new task. Figure 5.5 shows the episode success rates (bar chart) and the successful action rate (line plot). By fine-tuning **w**, the model quickly adapts to new tasks, yielding both high episode success rate and successful action rate. In contrast, the model trained from scratch takes substantially longer to converge. We also experiment with fine-tuning the entire model, and it suffers from similar slow convergence.

5.5.3 Learning Affordances

An agent in an interactive environment needs to be able to reason about the causal effects of actions. We expect our SR model to learn the pre- and post-conditions of actions through interaction, such that it develops a notion of affordance [75], i.e., which actions can be performed under a circumstance. In the real world, such knowledge could help prevent damages to the agent and the environment caused by unexpected or invalid actions.

We first evaluate each network's ability to *implicitly* learn affordances when trained on the tasks in Section 5.5.1. In these tasks, we penalize unnecessary actions with a small time penalty, but we do not explicitly tell the network which actions succeed and which fail. Figure 5.6 illustrates

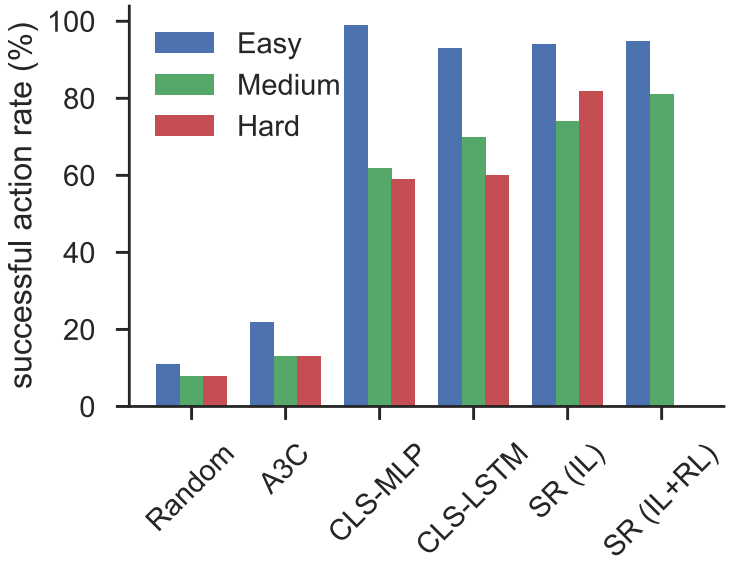


Figure 5.6: We compare the different models' likelihood of performing a successful action during execution. A3C suffers from the large action space due to naïve exploration. Imitation learning models are capable of differentiating between successful and unsuccessful actions because the supervised loss discourages the selection of unsuccessful actions.

that a standard reinforcement learning method cannot filter out unnecessary actions especially given delayed rewards. Imitation learning methods produce significantly fewer failed actions because they can directly evaluate whether each action gets them closer to the goal state.

We also analyze the successor network's capability of *explicitly* learning affordances. We train our SR model with reinforcement learning, by executing a completely random policy in the scene. We define the immediate reward of issuing a successful action as $+1.0$ and an unsuccessful one as -1.0. The agent learns in 10,000 episodes. Figure 5.7 shows a t-SNE [152] visualization of the state-action features $\phi_{s,a}$. We see that the network learns to cluster successful state action pairs (shown in green) separate from unsuccessful pairs (orange). The network achieves an ROC-AUC of 0.91 on predicting immediate rewards over random state-action actions, indicating that the model can differentiate successful and unsuccessful actions by performing actions and learning from their outcomes.

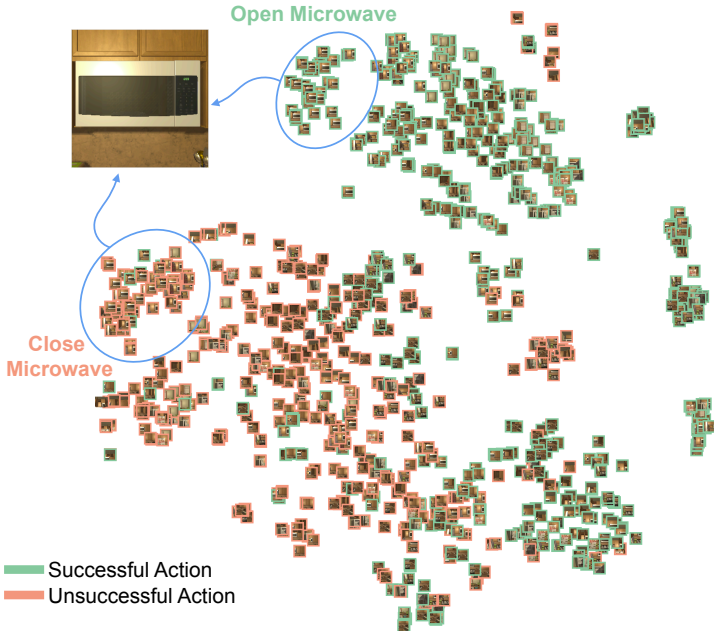


Figure 5.7: Visualization of a t-SNE [152] embedding of the state-action vector $\phi_{s,a}$ for a random set of state-action pairs. Successful state-action pairs are shown in green, and unsuccessful pairs in orange. The two blue circles highlight portions of the embedding with very similar images but different actions. The network can differentiate successful pairs from unsuccessful ones.

5.6 Discussions

In this chapter, we argue that visual semantic planning is an important next task in computer vision. Our proposed solution shows promising results in predicting a sequence of actions that change the current state of the visual world to a desired goal state. We have examined several different tasks with varying degrees of difficulty and show that our proposed model based on deep successor representations achieves near optimal results in the challenging THOR environment. We also show promising cross-task knowledge transfer results, a crucial component of any generalizable solution. Our qualitative results show that our learned successor features encode knowledge of object affordances, and action pre-conditions and post-effects.

Chapter 6

Neural Task Programming

6.1 Introduction

Autonomy in complex manipulation tasks, such as object sorting, assembly, and de-cluttering, requires sequential decision making with prolonged interactions between the robot and the environment. Planning for a complex task and, vitally, adapting to new task objectives and initial conditions is a long-standing challenge in robotics [27, 61].

Consider an object sorting task in a warehouse setting — it requires sorting, retrieval from storage, and packing for shipment. Each task is a sequence of primitives, such as `pick_up`, `move_to`, and `drop_into`, that can be composed into manipulation sub-tasks such as grasping and placing. This problem has an expansive space of variations, such as different objects-bin maps in sorting, permutations of sub-tasks, varying length order lists, resulting in a large space of tasks. As a concrete example, Figure 6.1(C) shows a simplified setup of the object sorting task. The task is to transport objects of four categories to four shipping containers. There is a total of 256 possible mappings between object categories and containers, and the variable number of object instances further increases the complexity. In this chapter, we attempt to address two challenges in complex task planning domains, namely (a) *learning policies that generalize to new task objectives*, and (b) *hierarchical composition of primitives for long-term environment interactions*.

We propose Neural Task Programming (NTP), a unified, task-agnostic learning algorithm that can be applied to a variety of tasks with latent hierarchical structure. The key underlying idea is to learn reusable representations shared across tasks and domains. NTP interprets a *task specification* (Figure 6.1 left) and instantiates a hierarchical policy as a neural program (Figure 6.1 middle), where the bottom-level programs are primitive actions that are executable in the environment. A task

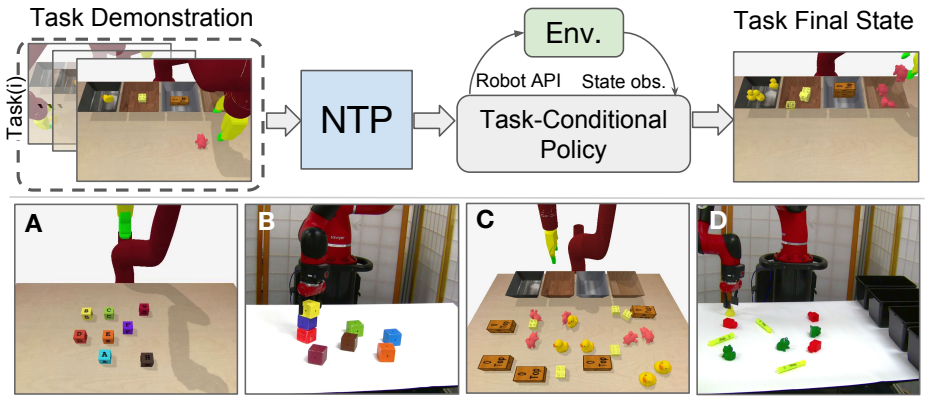


Figure 6.1: (Top) At test time, NTP instantiates a task-conditional policy (a neural program) that performs the specified task by interpreting a demonstration of a task. The policy interacts with the environment through robot APIs. (Bottom) We evaluate NTP on Block Stacking (A,B), Object Sorting (C, D) and Table Clean-up tasks in both simulated and real environment.

specification is defined as a time-series that describes the procedure and the final objective of a task. It can either be a task demonstration recorded as a state trajectory, first/third-person video demonstrations, or even a list of language instructions. In this work, we use *task demonstration* as the task specification. We experiment with two forms of task demonstration: location trajectories of objects that are involved in a task, and a third-person video demonstration of a task. NTP decodes the objective of a task from the input specification and factorizes it into sub-tasks, interacting with the environment with closed-loop feedback until the goal is achieved (Figure 6.1 right). Each program call takes as input the environment observation and a task specification, producing the next sub-program and a corresponding sub-task specification. The lowest level of the hierarchy is symbolic actions captured through a Robot API.

This hierarchical decomposition encourages information hiding and modularization, as lower-level modules only access their corresponding sub-task specifications that pertain to their functionality. It prevents the model from learning spurious dependencies on training data, resulting in better reusability. Essentially, NTP addresses the key challenges in task generalization: meta-learning for cross-task transfer and hierarchical model to scale to more complex tasks. Hence, NTP builds on the strengths of neural programming and hierarchical RL while compensating for their shortcomings.

We demonstrate that NTP generalizes to three kinds of variations in task structure: 1) *Task Length*: varying number of steps due to the increasing problem size (e.g., having more objects to transport); 2) *Task Topology*: the flexible permutations and combinations of sub-tasks to reach the same end goal (e.g., manipulating objects in different orders); and 3) *Task Semantics*: the varying

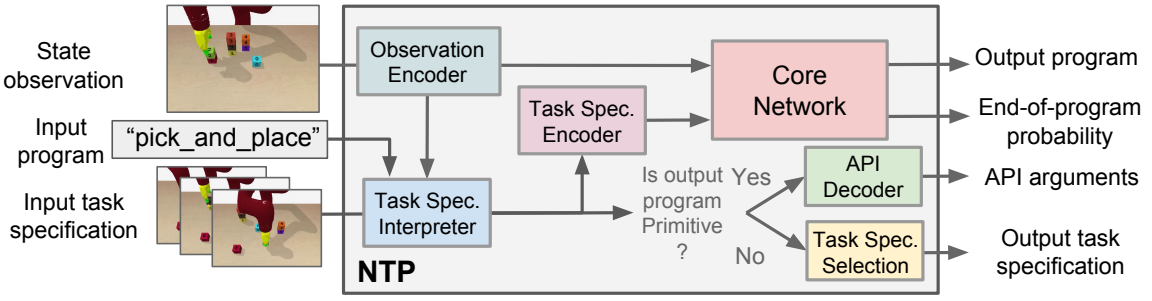


Figure 6.2: Neural Task Programming (NTP): Given an input program, a task specification, and the current environment observation, a NTP model predicts the sub-level program to run, the sub-sequence of the task specification that the sub-level program should take as input, and if the current program should stop.

task definitions and success conditions (e.g., placing objects into a different container).

We evaluate NTP on three table-top manipulation tasks that require long-term interactions: Block Stacking, Object Sorting, and Table Clean-up. We evaluate each task in both simulated and real-robot setups.

Summary of Contributions:

1. Our primary contribution is a novel modeling framework: NTP that enables meta-learning on hierarchical tasks.
2. We show that NTP enables knowledge transfer and one-shot demonstration based generalization to novel tasks with increasing lengths, varying topology, and changing semantics without restriction on initial configurations.
3. We also demonstrate that NTP can be trained with visual input (images and video) end-to-end.

6.2 Related work

Skill Learning. The first challenge is learning policies that adapt to new task objectives. For learning a single task policy, traditional methods often segment a complex task into hand-engineered state machine composed of motion primitives [27, 61, 215]. Although the model-based approaches are well-founded in principle, they require meticulous model specification and task-specific treatment leading to challenges in scaling. Contrarily, learning-based methods such as reinforcement learning (RL) have yielded promising results using end-to-end policy learning that obviates the need for manually designed state representations through data-driven task-salient features [161, 271]. Yet these methods fall short because they need task-specific reward functions [166].

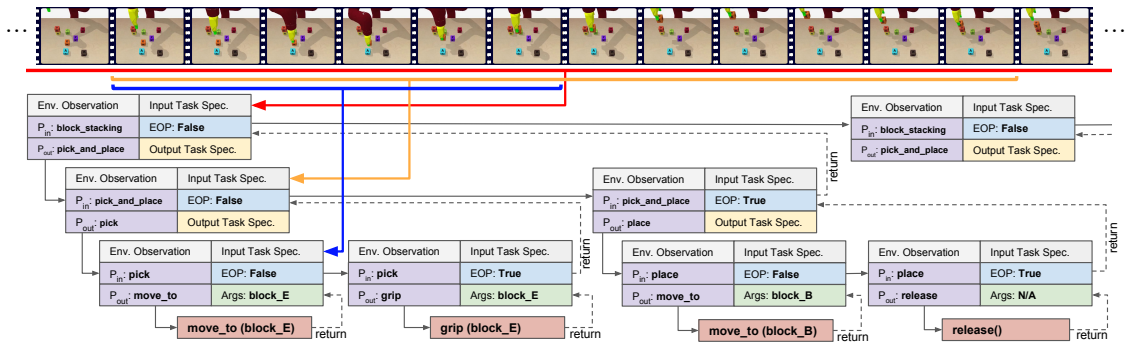


Figure 6.3: Sample execution trace of NTP on a block stacking task. The task is to stack lettered blocks into a specified configuration (`block_D` on top of `block_E`, `block_B` on top of `block_D`, etc). Top-level program `block_stacking` takes in the entire demonstration as input (red window), and predicts the next sub-program to run is `pick_and_place`, and it should take the part of task specification marked by the orange window as the input specification. The bottom-level API call moves the robot and close / open the gripper. When End of Program (EOP) is True, the current program stops and return its caller program.

Learning from Demonstrations. LfD fills these gaps by avoiding the need to define state machines or reward functions. The objective in LfD is to learn policies that generalize beyond the provided examples and are robust to perturbations [10, 119] A common treatment to LfD is to model data as samples from an expert policy for a fixed task, and use behavior cloning [104, 201] or reward function approximation [167] to output an expert-like policy for that task. However, learning policies that generalize to new objectives with LfD remains largely an unexplored problem.

Few-Shot Generalization in LfD. Our work is an instantiation of the decades-old idea of meta-learning with few examples [60, 247]. It has seen a recent revival in deep learning in part because it can address the problems above [248]. Our setting resembles learning by demonstration (LfD) in robotics [19], particularly one-shot imitation [52, 258]. Our method *learns to learn* from an input task specification during training. At test time, it generates a policy conditioned on a *single* demonstration provided as a time-series showing the task execution. While similar in these aspects, existing works in both skill learning and LfD are inept at tasks with sparse reward functions and complex hierarchical structures such as Montezuma's Revenge [129].

Hierarchical Skill Composition. The second challenge we consider is the hierarchical composition of primitives to enable long-term robot-environment interaction. The idea of using hierarchical models for complex tasks has been widely explored in both reinforcement learning and

robotics [119, 230]. A common treatment to manage task structure complexity is to impose hierarchy onto the learned policy. The *options* framework composes primitive actions into multi-step actions, which facilitates policy learning at higher-level semantic and/or temporal abstraction [67, 232]. Notable examples include structured reinforcement learning methods, especially hierarchical variants of RL that handle decomposition through multi-stage policies operating over options [12, 129, 179]. However, the nave use of a hierarchical RL model with "sub-policies" or options optimized for a specific task doesn't guarantee modularity or reusability across task objectives.

The core idea of NTP resonates with recent works on dynamic neural networks, which aim to learn and reuse primitive network modules. These methods have been successfully applied to several domains such as robot control [7] and visual question answering [8]. However, they have exhibited limited generalization ability across tasks. In contrast, we approach the problem of hierarchical task learning via neural programming to attain modularization and reusability [194]. As a result, our model achieves significantly better generalization results than non-hierarchical models such as [52].

FSMs and Neural Program Induction. An exciting and non-intuitive insight of this chapter is that the well-studied Finite State Machine (FSM) model lends itself to learning reusable hierarchical policies thereby addressing the problem of composability without the need for hand-crafting state transitions. There have been a few studies learning FSMs from data [76, 127]. In line with the idea, recent works in neural programming using deep models have enabled symbolic reasoning systems to be trained end-to-end, which have shown potential to handle multi-modal and raw input/out data [49, 194] and achieve symbolic generalization [28].

NTP belongs to a family of neural program induction methods, where the goal is to learn a latent program representation that generates program outputs [49, 194]. While these models have been shown to generalize on task length, they are tested on basic computational tasks only with limited generalization to task semantics and topology. Similar to NTP, Neural Programmer-Interpreter (NPI) [194] has proposed to use a task-agnostic recurrent neural network to represent and execute programs. In contrast to previous work on neural program induction, NPI-based models are trained with richer supervision from the full program execution traces and can learn semantically meaningful programs with high data efficiency. However, program induction, including NPI, is not capable of generalizing to novel programs without training.

NTP is a meta-learning algorithm that learns to instantiate *neural programs* given demonstrations of tasks, thereby generalizing to unseen tasks/programs. Intuitively, NTP decomposes the overall objective (e.g., object sorting) into simpler objectives (e.g., pick and place) recursively. For

each of such sub-tasks, NTP delegates a neural program to perform the task. The neural programs, together with the task decomposition mechanism, are trained end-to-end.

While previous work has largely focused on executing a pre-defined task one at a time NTP not only exhibits one-shot generalization to tasks with longer lengths as NPI, but also generalizes to sub-task permutations (topology) and success conditions (semantics).

6.3 Problem Formulation

We consider the problem of an agent performing actions to interact with an environment to accomplish tasks. Let $\mathbb{T}$ be the set of all tasks, $\mathbb{S}$ be the environment state space, and $\mathbb{A}$ be the action space. For each task $t \in \mathbb{T}$, the Boolean function $g : \mathbb{S} \times \mathbb{T} \rightarrow \{0,1\}$ defines the success condition of the task. Given any state $s \in \mathbb{S}$, $g(s,t) = 1$ if the task t is completed in the state s, and $g(s,t) = 0$ otherwise. The task space $\mathbb{T}$ can be infinite. We thus need a versatile way to describe the task semantics. We describe each task using a task specification $\psi(t) \in \Psi$, where Ψ is the set of all possible task specifications. Formally, we consider a task specification as a sequence of random variables $\psi(t) = \{x_1, x_2, \ldots, x_N\}$.

NTP takes a *task specification* $\psi(t)$ as input in order to instantiate a policy. $\psi(t)$ is defined as a time series that describes the procedure and the final objective of the task. In experiments, we consider two forms of task specifications: trajectories of object locations and raw video sequences. In many real-world tasks, the agent has no access to the underlying environment states. It only receives a sample of environment observation $o \in \mathbb{O}$ that corresponds to the state s, where $\mathbb{O}$ is the observation space. Our goal is to learn a "meta-policy" that instantiates a feedback policy from a specification of a task, $\tilde{\pi} : \Psi \rightarrow (\mathbb{O} \rightarrow \mathbb{A})$. At test time, a specification of a new task $\psi(t)$ is input to NTP. The meta-policy then generates a policy $\pi(a|o; \psi(t)) : \mathbb{O} \rightarrow \mathbb{A}$, to reach task-completion state s_T where $g(s_T, t) = 1$.

Why use Neural Programming for LfD? Previous work has mostly used a monolithic network architecture to represent a goal-driven policy [52, 210, 271]. These methods cannot exploit the compositional task structures to facilitate modularization and reusability. Instead, we represent our policy $\tilde{\pi}$ as a neural program that takes a task specification as its input argument. As illustrated in Figure 6.2, NTP uses a task-agnostic core network to decide which sub-program to run next and adaptively feeds a subset of the task specification to the next program. Intuitively, NTP recursively decomposes a task specification and solves a hierarchical task by divide-and-conquer. Figure 6.3 highlights this feature with a sample execution of a task. Our method extends upon a special type

of neural programming architecture named Neural Programmer-Interpreter (NPI) [28, 194]. NPI generalizes well to input size but cannot generalize to unseen task objectives. NTP combines the idea of meta-learning and NPI. The ability to interpret task specifications and instantiate policies accordingly makes NTP generalize across tasks.

Neural Programmer-Interpreter. Before introducing our NTP model, it is useful to briefly overview the NPI paradigm [194]. NPI is a type of neural program induction algorithm, in which a network is trained to imitate the behavior of a computer program, i.e., the network learns to invoke programs recursively given certain context or stop the current program and return to upper-level programs. The core of NPI is a long-short memory (LSTM) [96] network. At the i-th time step, it selects the next program to run conditioned on the current observation o_i and the previous LSTM hidden units h_{i-1}. A domain-specific encoder is used to encode the observation o_i into a state representation s_i. The NPI controller takes as input the state s_i, the program embedding p_i retrieved from a learnable key-value memory structure $[M^{key}; M^{prog}]$, and the current arguments a_i. It generates a program key, which is used to invoke a sub-program p_{i+1} using content-based addressing, the arguments to the next program a_{i+1}, and the end-of-program probability r_i. The NPI model maintains a program call stack. Each time a sub-program is called, the caller's LSTM hidden units embedding and its program embedding is pushed to the stack. Formally, the NPI core has three learnable components, a domain-specific encoder f_{enc}, an LSTM f_{lstm}, and an output decoder f_{dec}. The full update being: $s_i = f_{enc}(o_i, a_i) \quad h_i = f_{lstm}(s_i, p_i, h_{i-1}) \quad r_i, p_{i+1}, a_{i+1} = f_{dec}(h_i)$. When executing a program with the NPI controller, it performs one of the following three things: 1) when the end-of-program probability exceeds a threshold α (set to 0.5), this program is popped up from the stack and control is returned to the called; 2) when the program is not primitive, a sub-program with its arguments is called; and 3) when the program is primitive, a low-level basic action is performed in the environments. The LSTM core is shared across all tasks.

6.4 Model

Overview. NTP has three key components: Task Specification Interpreter f_{TSI}, Task Specification Encoder f_{TSE}, and a core network f_{CN} (Figure 6.2. The Task Specification Encoder transforms a task specification ψ_i into a vector space. The core network takes as input the state s_i, the program p_i, and the task specification ψ_i, producing the next sub-program to invoke p_{t+1} and an end-of-program probability r_t. The program returns to the caller when r_t exceeds a threshold α (set to 0.5). We detail the inference procedure in Algorithm 1.

NTP vs NPI: We highlight three main differences of NTP than the original NPI: (1) NTP can interpret task specifications and perform hierarchical decomposition and thus can be considered as a meta-policy; (2) it uses APIs as the primitive actions to scale up neural programs for complex tasks; and (3) it uses a reactive core network instead of a recurrent network, making the model less history-dependent, enabling feedback control for recovery from failures. In addition to the three key components, NTP implements two modules similar to the NPI architectures [28, 194]: (1) domain-specific task encoders that map an observation to a state representation $s_i = f_{ENC}(o_i)$, and (2) a key-value memory that stores and retrieves embeddings: $j^* = \arg\max_{j=1...N}(M^{key}_{j,:} k_i)$ and $p_i = M^{prog}_{j^*,:}$, where k_i is the program key predicted by the core network.

Scaling up NTP with APIs. The bottom-level programs in NPI correspond to primitive actions that are executable in the environment. To scale up neural programs in coping with the complexity of real-world tasks, it is desirable to use existing tools and subroutines (i.e., motion planner) such that learning can be done at an abstract level. In computer programming, application programming interfaces (APIs) have been a standard protocol for developing software by using basic modules. Here we introduce the concept of API to neural programming, where the bottom-level programs correspond to a set of robot APIs, e.g., moving the robot arm using inverse kinematics. Each API takes specific arguments, e.g., an object category or the end effector's target position. NTP jointly learns to select APIs functions and to generate their input arguments. The APIs that are used in the experiments are `move_to`, `grip`, and `release`. `move_to` takes an object index as the API argument and calls external functions to move the gripper to above the object whose position is either given by the simulator or predicted by an object detector. `grip` closes the gripper and `release` opens the gripper.

Task Specification Interpreter. The Task Specification Interpreter, taking a task specification as input, chooses to perform one of the two operations: (1) when the current program p is not primitive, it predicts the sub-task specification for the next sub-program; and (2) when p is primitive (i.e., an API), it predicts the arguments of the API.

Let ψ_i be the task specification of the i-th program call, where ψ_i is a sequence of random variables $\psi_i = \{x_1, x_2, \ldots, x_{N_i}\}$. The next task specification ψ_{i+1} is determined by three inputs: the environment state s_i, the current program p_i, and the current specification ψ_i. When p_i is a primitive, TSI uses an API-specific decoder (i.e., an MLP) to predict the API arguments from the tuple (s_i, p_i, ψ_i).

We focus on the cases when p_i is not primitive. In this case, TSI needs to predict a sub-task specification ψ_{i+1} for the next program p_{i+1}. This sub-task specification should only access

Algorithm 1 NTP Inference Procedure

Inputs: task specification ψ, program id i, and environment observation o
function RUN(i, ψ)
  $r \leftarrow 0$, $p \leftarrow M^{prog}_{i,:}$, $s \leftarrow f_{ENC}(o)$, $c \leftarrow f_{TSE}(\psi)$
  while $r < \alpha$ **do**
    $k, r \leftarrow f_{CN}(c, p, s)$, $\psi_2 \leftarrow f_{TSI}(\psi, p, s)$
    $i_2 \leftarrow \arg\max_{j=1...N}(M^{key}_{j,:}k)$
    if program i_2 is primitive **then** ▷ if i_2 is an API
      $\mathbf{a} \leftarrow f_{TSI}(\psi_2, i_2, s)$ ▷ decode API args
      RUN_API($i_2, \mathbf{a}$) ▷ run API i_2 with args $\mathbf{a}$
    else
      RUN(i_2, ψ_2) ▷ run program i_2 w/ task spec ψ_2
    end if
  end while
end function

relevant information to the sub-task. To encourage information hiding from high-level to low-level programs, we enforce the scoping constraint, such that ψ_{i+1} is a contiguous subsequence of ψ_i. Formally, given $\psi_i = \{x_1, x_2, \ldots, x_{N_i}\}$, the goal is to find the optimal contiguous subsequence $\psi_{i+1} = \{x_p, x_{p+1}, \ldots, x_{q-1}, x_q\}$, where $1 \leq p \leq q \leq N_i$.

Subsequence Selection (Scoping). We use a convolutional architecture to tackle the subsequence selection problem. First, we embed each input element $\psi_i = \{x_1, x_2, \ldots, x_{N_i}\}$ into a vector space $\phi_i = \{w_1, w_2, \ldots, w_{N_i}\}$, where each $w_i \in \mathbb{R}^d$. We perform temporal convolution at every temporal location j of the sequence ϕ_i, where each convolutional kernel is parameterized by $W \in \mathbb{R}^{m \times dk}$ and $b \in \mathbb{R}^m$, which takes a concatenation of k consecutive input elements and produces a single output $y^j_i \in \mathbb{R}^m$. We use *relu* as the nonlinearities. The outputs from all convolutional kernels $\mathbf{y}^j_i$ are concatenated with the program embedding p_i and the encoded states s_i into a single vector $h_j = [p_i; \mathbf{y}^j_i; s_i]$. Finally, we compute the softmax probability of four scoping labels $\Pr_j(l \in \{\texttt{Start}, \texttt{End}, \texttt{Inside}, \texttt{Outside}\})$. These scoping labels indicate whether this temporal location is the start/end point of the correct subsequence, or if it resides inside/outside the subsequence. We use these probabilities to decode the optimal subsequence as the output sub-task specification ψ_{i+1}.

The decoding process can be formulated as the maximum contiguous subsequence sum problem, which can be solved optimally in linear time. However in practice, taking the start and end points with the highest probabilities results in a good performance. In our experiments, we set $\psi_{i+1} = \{x_{\text{st}}, x_{\text{st}+1}, \ldots, x_{\text{ed}}\}$, where $st = \arg\max_{j=1...N_i} \Pr_j(\texttt{Start})$ and $ed = \arg\max_{j=1...N_i} \Pr_j(\texttt{End})$. This

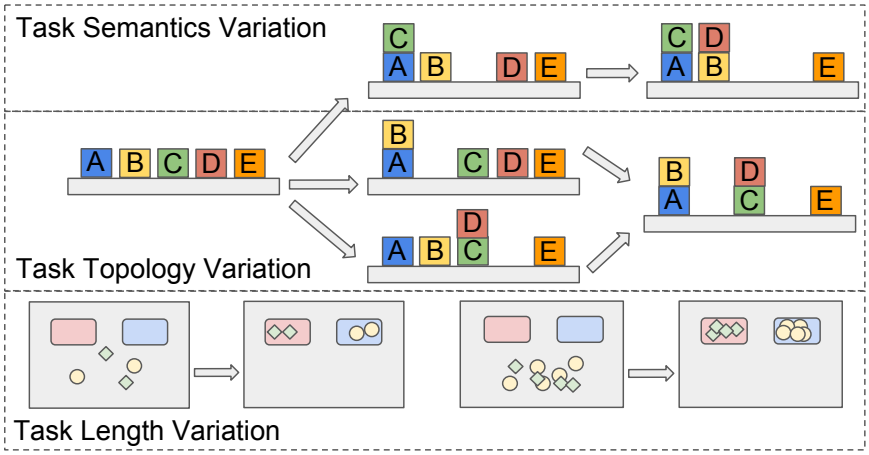


Figure 6.4: The variability of a task structure consists of changing success conditions (task semantics), variable subtask permutations (task topology), and larger task sizes (task length). We evaluate the ability of our proposed model in generalizing towards these three types of variations.

process is illustrated in Figure 6.3, wherein the model factorizes a video sequence which illustrates the procedure of *pick_and_place* into a fraction that only illustrates *pick*. This convolutional TSI architecture is invoked recursively along the program execution trace. It decomposes a long task specification into increasingly fine-grained pieces from high-level to low-level tasks. This method naturally enforces the scoping constraint. Our experimental results show that such information hiding mechanism is crucial to good generalization.

Model Training. We train the model using rich supervision from program execution traces. Each execution trace is a list of tuples $\{\xi_t \,|\, \xi_t = (\psi_t, p_t, s_t), t = 1 \ldots T\}$, where T is the length of the execution trace. Our training objective is to maximize the probability of the correct executions over all the tasks in the dataset $\mathscr{D} = \{(\xi_t, \xi_{t+1})\}$, such that $\theta^* = \Sigma_{\mathscr{D}} \log \Pr[\xi_{t+1} | \xi_t; \theta]$.

We collect a dataset that consists of execution traces from multiple types of tasks and their task specifications. For each specification, we provide the ground-truth hierarchical decomposition of the specification for training by rolling a hard-coded expert policy. We use cross-entropy loss at every temporal location of the task specification to supervise the scoping labels. We also adopted the idea of adaptive curriculum from NPI [194], where the frequency of each mini-batch being fetched is proportional to the model's prediction error with respect to the corresponding program.

6.5 Experiments

The goal of our experimental evaluation is to answer the following questions: (1) Does NTP generalize to changes in all three dimensions of variation: length, topology, and semantics, as illustrated in Figure 6.4, (2) Can NTP use image-based input without access to ground truth state, and (3) Would NTP also work in real-world tasks which have combinations of these variations. We evaluate NTP in three robot manipulation tasks: Object Sorting, Block Stacking, and Table Clean-up. Each of these tasks requires multiple steps to complete and can be recursively decomposed into repetitive sub-tasks.

Input State Representation. We use an expert policy to generate program execution traces as training data. An expert policy is an agent with hard-coded rules that call programs (`move_to`, `pick_and_drop`, etc.) to perform a task. In our experiment, we use the demonstration of a robot carrying out a task as the task specification. For all experiments, unless specified, the state representation in the task demonstrations is in the form of object position trajectories relative to the gripper frame. In the Block Stacking experiment, we also report the results of using a learned object detector to predict object locations and the results of directly using RGB video sequence as state observations and task demonstrations.

Simulator Setup. We conduct our experiments in a 3D environment simulated using the Bullet Physics engine [38]. We use a disembodied PR2 gripper for both gathering training data and evaluation. We also evaluate NTP on a simulated 7-DoF Sawyer arm with a parallel-jaw gripper as shown in Figure 6.1. Since NTP only considers end-effector pose, the choice of robot does not affect its performance in the simulated environment.

Real-Robot Setup. We also demonstrate NTP's performance on the Block Stacking and the Object Sorting tasks on a 7-DoF Sawyer arm using position control. We use NTP models that are trained with simulated data. Task demonstration are obtained in the simulator, and the instantiated NTP models are executed on the robot. All real-robot experiments use object locations relative to the gripper as state observations. A Kinect2 camera is used to localize objects in the 3D scene.

Evaluation Metrics. We evaluate NTP on three variations of task structure as illustrated in Figure 6.4: 1) *task length*: varying number of steps due to the increasing problem size (e.g., having more objects to transport); 2) *task topology*: variations in permutations of steps of sub-tasks to reach the same end goal (e.g., manipulating objects in different orders); and 3) *task semantics*: the unseen task objectives and success conditions (e.g., placing objects into a different container).

We evaluate *Task length* on the Object Sorting task varying the number of objects instances from

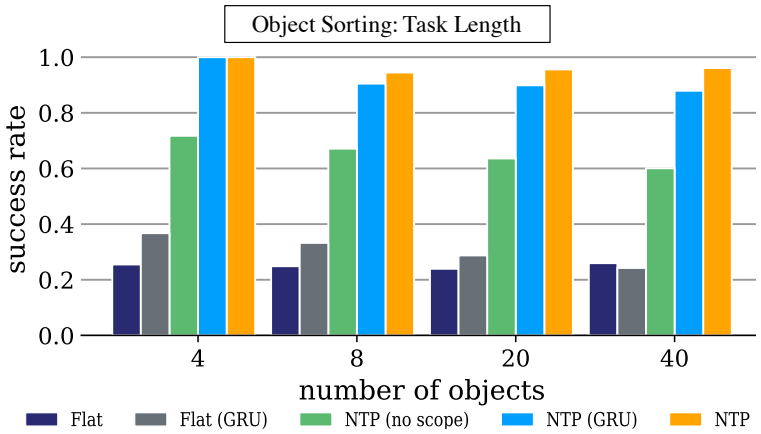


Figure 6.5: Task length: Evaluation of the Object Sorting in simulation. The axes represent mean success rate (y) with 100 evaluations each and the number of objects in unseen task instances (x). NTP generalizes to increasingly longer tasks while baselines do not.

1 to 10 per category. Further, we evaluate *Task Topology* on the Block Stacking task with different permutations of pick-and-place sub-tasks that lead to the same block configurations. Finally, we evaluate *Task Semantics* on Block Stacking on a held-out set of task demonstrations that lead to unseen block configurations as task objectives.We report success rates for simulation tasks, and we analyze success rates, causes of failure, and proportion of task completed for real-robot evaluation. All objects are randomly placed initially in all of the evaluation tasks in both the simulated and the real-robot setting.

Baselines. We compare NTP to four baselines architecture variations. (1) **Flat** is a non-hierarchical model, similar to [52], that takes as input task demonstration and current observation, and directly predicts the primitive APIs instead of calling hierarchical programs. (2) **Flat (GRU)** is the Flat model with a GRU cell. (3) **NTP (no scope)** is a variant of the NTP model that feeds the entire demonstration to the subprograms, thereby discarding the scoping constraint. (4) **NTP (GRU)** is a complete NTP model with a GRU cell. This is to demonstrate that the reactive core network in NTP can better generalize to longer tasks and recover from unexpected failures due to noise, which is crucial in robot manipulation tasks.

6.5.1 Experiment 1: Object Sorting

Setup. The goal of Object Sorting is to transport objects randomly scattered on a tabletop into their respective shipping containers stated in the task demonstration. We use 4 object categories

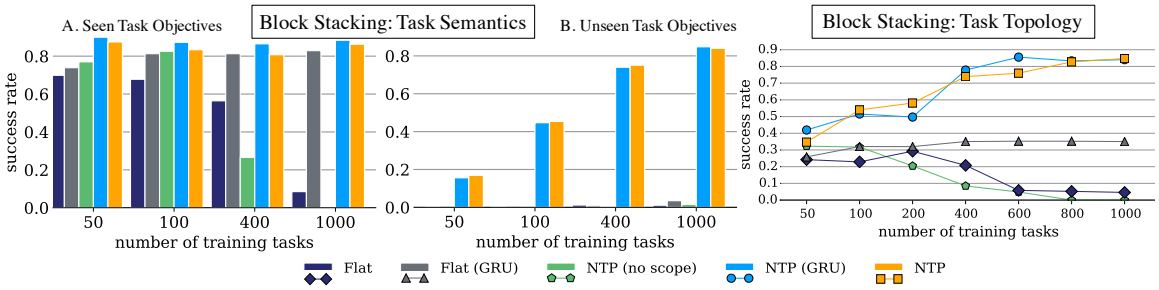


Figure 6.6: (Left) Task semantics: Simulated evaluation of the Block Stacking. The x-axis is the number of tasks used for training. The y-axis is the overall success rate. (A) and (B) show that NTP and its variants generalize better to novel task demonstrations and objectives as the number of training tasks increases. (Right) Task topology: Simulated evaluation of the Block Stacking. NTP shows better performance in task topology generalization as the number of training tasks grows. In contrast, the flat baselines cannot handle topology variability.

and 4 containers in evaluating the Object Sorting task. In the real robot setup, a toy duck, toy frog, lego block, and marker are used as the objects for sorting, and are sorted into 4 black plastic bins. This results in a total of $4^4 = 256$ category-container combinations (multiple categories may be mapped to the same container). However, as each category can be mapped to 4 possible containers, a minimum of 4 tasks can cover all possible category-container pairs. We select these 4 tasks for training and the other 252 unseen tasks for evaluation. We train all models with 500 trajectories. Each test run is on 100 randomly-selected unseen tasks.

Simulator. As shown in Figure 6.5, NTP significantly outperforms the flat baselines. We examine how the task size affects its performance. We vary the numbers of objects to be transported from 4 to 40 in the experiments. The result shows that NTP retains a stable and good performance (over 90%) in longer tasks. On the contrary, the flat models' performances decline from around 40% to around 25%, which is close to random. The performance of the NTP (GRU) model also declines faster comparing to the NTP model as the number of objects increases. This comparison illustrates NTP's ability to generalize towards task length variations.

Real robot. Table 6.1 shows the results of the Object Sorting task on the robot. We use 4 object categories with 3 instances of each category. We carried out a total of 10 evaluation trials on randomly selected unseen Object Sorting tasks. 8 trials completed successfully, and 2 failed due to of manipulation failures: a grasp failure and a collision checking failure.

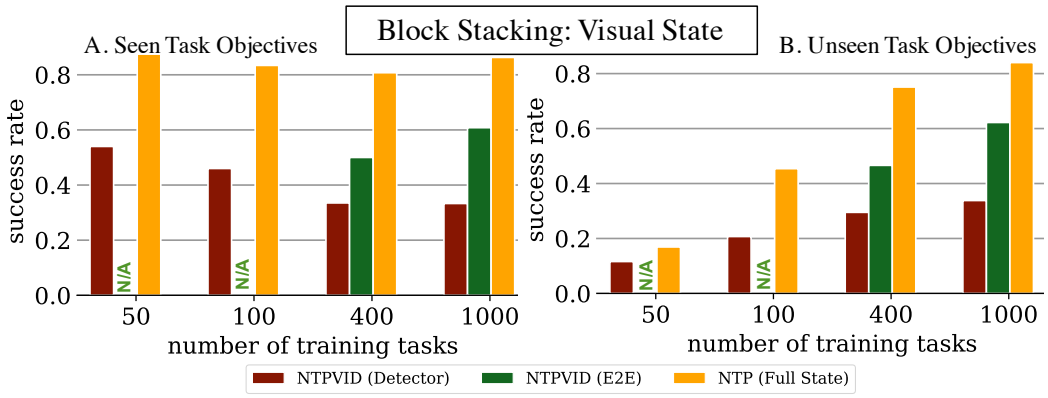


Figure 6.7: NTP with Visual State: NTPVID (Detector) uses an object detector on images which is subsequently used as state in NTP. NTP (E2E) is an end-to-end model trained completely on images with no low-level state information. We note that in the partial observation case (only video), similar learning trends were observed as compared to fully observed case (NTP (Full State)), albeit with a decrease in performance.

6.5.2 Experiment 2: Block Stacking

Setup. The goal of Block Stacking is to stack a set of blocks into a target configuration, similar to the setup in [52]. We use 8, 5×5 cm wooden cubes of different colors both in simulation and with real-robot. We randomly generate 2000 distinct Block Stacking task instances. Two tasks are considered equivalent if they have the same end configuration. We use a maximum of 1000 training tasks and 100 trials for each task, leaving the remaining 1000 task instances as unseen test cases. A task is considered successful if the end configuration of the blocks matches the task demonstration. We evaluate both seen and unseen tasks, i.e., whether the end configuration appears in training set. We use $N = 8$ blocks in our evaluation.

Simulator. Figure 6.6 shows that all models except the Flat baseline are able to complete the seen tasks at around 85% success rate. The performance of the Flat baseline decreases dramatically when training with more than 400 tasks. It is because the Flat model has very limited expressiveness power to represent complex tasks. The Flat (GRU) model performs surprisingly well on the seen tasks. However, as shown in Figure 6.6, both Flat and Flat (GRU) fail to generalize to unseen tasks. We hypothesize that the Flat (GRU) baseline simply memorizes the training sequences. On the other hand, NTP achieves increasingly better performances when the diversity of the training data increases.

We evaluate task topology generalization on random permutations of the pick-and-place subtasks that lead to the same end configuration. Specifically, the task variations are generated by randomly shuffling the order that the "block towers" are built in the training tasks. Figure 6.6 illustrates that NTP generalizes better towards variable topologies when trained on a larger variety of tasks. We find that increasing the diversity of training data facilitates NTP to learn better generalizable modules.

Next, we evaluate task semantics generalization. The variability of real-world environments prevents any task-specific policy learning method from training for every possible task. Figure 6.6(A) illustrates that NTP generalizes well to novel task demonstrations and new goals. As the number of training tasks increases, both NTP and its recurrent variation steadily improve their performance on the unseen tasks. When trained with 1000 tasks, their performances on unseen tasks are almost on par with that of seen tasks.

The performance gaps between NTP (no scope) and NTP highlight the benefit of the scoping constraint. NTP (no scope) performance drops gradually as the task size grows implying that the programs in NTP learn modularized and reusable semantics due to information hiding, which is crucial to achieving generalization towards new tasks.

Real robot. Table 6.1 shows the results of the Object Sorting task in the real world setting. We carried out 20 trails of randomly selected unseen Block Stacking tasks. Out of the 2 failure cases, one is caused by an incorrect placing; the other is caused by the gripper knocking down a stacked tower and not able to recover from the error.

Adversarial dynamics. We show that the reactive core network in NTP enables it to better recover from failures compared to its recurrent variation. We demonstrate this by performing Block Stacking under an adversary. Upon stacking each block, an adversary applies a force to the towers with a probability of 25%. The force can knock down the towers. We evaluate NTP and its recurrent variant on the 1000 unseen tasks. Table 6.2 shows that under the same adversary, the success rate of NTP with the GRU core decreases by 46%, whereas the success rate of NTP only decreases by 20%. This indicates that a reactive model is more robust against unexpected failures as its behavior is less history dependent than the recurrent counterpart. We also demonstrate this feature in the supplementary video in the real world setting.

NTP with visual state. This experiment examines the ability of NTP to learn when demonstrations come in the form of videos and the state is a single image. Unlike the full state information used in experiments thus far, we train an NTP model NTPVID (E2E) to jointly learn a policy and task-relevant features without explicit auxiliary supervision. An alternative is to use a 2-phase pipeline

Table 6.1: Real robot evaluation. Results of 20 unseen Block Stacking evaluations and 10 unseen sorting evaluations on Sawyer robot for the NTP model trained on simulator. NTP Fail denotes an algorithmic mistake, while Manipulation Fail denotes a mistake in physical interaction (e.g., grasping failures and collisions).

Tasks	# Trials	Success	NTP Fail	Manipulation Fail
Block Stacking	20	0.9	0.05	0.05
Sorting	10	0.8	0	0.20

Table 6.2: Adversarial Dynamics. Evaluation results of the Block Stacking Task in a simulated adversarial environment. We find that NTP with GRU performs markedly worse with intermittent failures.

Model	No failure	With failures
NTP	0.863	0.663
NTP (GRU)	0.884	0.422

with an object detector as state preprocessor for NTP, termed as NTPVID (Detector). The detector is a separately trained CNN to predict object position in $\mathbb{R}^3$.

We explore these results in Figure 6.7, where we see compare the visual models (NTPVID (E2E) and NTPVID (Detector)) against the best full state model (NTP), all trained on 100 demonstrations per task, for a varying number of tasks. For NTPVID (E2E) we use a 7-layer convolutional network, which takes as input a 64×64 image and outputs a length 128 feature vector. For NTPVID (Detector) , we use a VGG16 based architecture, predicting the position of the N-task objects from an input image of size 224×224.

We note that NTPVID (E2E) outperforms NTPVID (Detector) and achieves a higher success rate despite only having partial state information. Both of these methods are inferior to the full-state NTP version. NTPVID (Detector) does not generalize due to task-specific state representation, and cascading errors in detection propagate to NTP reducing performance even when using a very deep network for the detection. The detector errors are Gaussian with standard deviation of 2 cm. However, this performance comes at a computational cost. NTPVID (E2E) was trained on 1000 training tasks for 10 days on 8 Nvidia Titan X GPUs. NTPVID (Detector) was trained for 24 hours on a single GPU. Due to computational cost, we only evaluated NTPVID (E2E) on 400 and 1000 training tasks.

6.5.3 Experiment 3: Table Clean-up

Setup. We also evaluate NTP on the Table Clean-up task, which exemplifies a practical real-world task. Specifically, the goal of the task is to clear up to 4 white plastic bowls and 20 red plastic forks into a bin such that the resulting stack of bowls and forks can be steadily carried away in a tray. Task variation comes in task length, where the number of utensils varies, and task topology, where the ordering in which bowls are stacked can vary. Using trajectories as demonstrations and object positions as state space, a model is trained using 1000 task instances.

Simulator. We observe that performance varies between 55%-100% where increasing errors with more objects are attributed to failures in collision checking, not incorrect decisions from NTP. The result shows that NTP retains its generalization ability in a task that requires multiple dimensions of generalization.

Real robot. We have also transferred the trained model on the real-Sawyer arm to evaluate the feasibility. Qualitative videos can be viewed from https://stanfordvl.github.io/ntp/.

6.6 Discussions

We introduced Neural Task Programming (NTP), a meta-learning framework that learns modular and reusable neural programs for hierarchical tasks. We demonstrate NTP's strengths in three robot manipulation tasks that require prolonged and complex interactions with the environment. NTP achieves generalization towards task length, topology, and semantics. This work opens up the opportunity to use generalizable neural programs for modeling hierarchical tasks. Future directions include 1) improving the state encoder to extract more task-salient information such as object relationships, 2) devising a richer set of APIs such as velocity and torque-based controllers, and 3) extending this framework to tackle more complex tasks on real robots.

Chapter 7

Neural Task Graph

7.1 Introduction

Learning sequential decisions and adapting to new task objectives at test time is a long-standing challenge in AI [27, 61]. In rich real domains, an autonomous agent has to acquire new skills with minimal supervision. Recent works have tackled the problem of one-shot imitation learning [52, 65, 258, 260] that learns from a single demonstration. In this chapter, we push a step further to address one-shot *visual* imitation learning that operates directly on *videos*. We first train a model on a set of seen in-domain tasks. The model can then be applied on a single video demonstration to obtain an execution policy of the new unseen task.

Learning directly from video is crucial for advancing the existing imitation learning approaches to real-world scenarios as it is infeasible to annotate states, such as object trajectories, in each video. We focus on *long-horizon tasks*, as real-world tasks such as cooking or assembly are inherently long-horizon and hierarchical. Recent works have attempted learning from pixel space [65, 150, 217, 264], but learning long-horizon tasks from video in a one-shot setting remains a challenge, since both the visual learning and task complexity exacerbate the demand for better data efficiency.

Our solution explicitly models the compositionality in the task structure and policy, enabling us to scale one-shot visual imitation to complex tasks. This is in contrast to previous works using unstructured task representations and policies [52, 65]. The use of compositionality has led to better generalization in Visual Question Answering [97, 106, 124] and Policy Learning [7, 48, 250]. We propose Neural Task Graph (NTG) Networks, a novel framework that uses task graph as the intermediate representation to explicitly modularize both the visual demonstration and the derived policy. NTG consists of a generator and an execution engine, where the generator builds a *task graph*

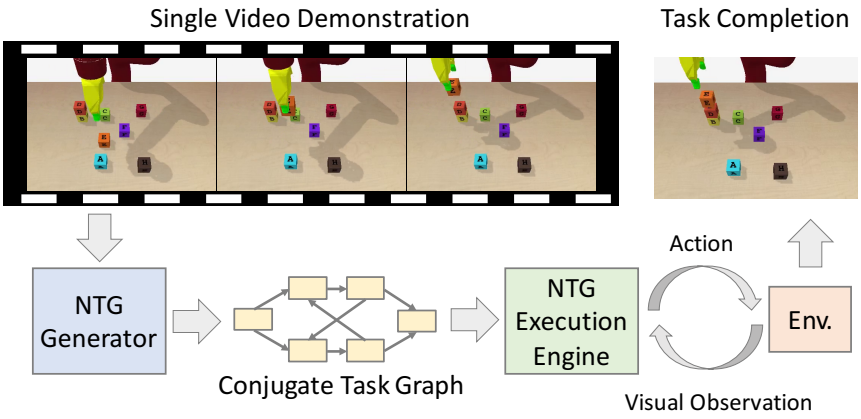


Figure 7.1: Our goal is to execute an unseen task from a single video demonstration. We propose Neural Task Graph Networks that leverage compositionality by using the task graph as the intermediate representation. This leads to strong inter-task generalization.

from the task demo video to capture the structure of the task, and the execution engine interacts with the environment to perform the task conditioned on the inferred task graph. Figure 7.1 shows an overview of NTG Networks.

The main technical challenge in using graphical task representations is that the unseen demos can easily introduce states that are never observed during training. For example, the goal state of an unseen block stacking task [52, 260] is a block configuration that never appears during training. This challenge is amplified by our goal of learning from visual observation without strong supervision, which obscures the state structure and prevents direct state space decomposition, as done in prior work [52]. Our key observation is that, while there can be countless possible states, the number of possible actions in a certain domain is often limited. We leverage this conjugate relationship between states and actions, and propose to learn NTG on the Conjugate Task Graph (CTG) [89], where the nodes are actions, and the states are captured by the edges. This allows us to modularize the policy and address the challenge of an unknown number of novel states. This is critical when operating in visual space, where states are high dimensional images and modeling a graph over a combinatorial state space is infeasible. Additionally, the CTG intermediate representation can yield alternate action sequences to complete the task, a property that is vital for generalization to unseen scenarios in a world with stochastic dynamics. This sets NTG apart from previous works that directly output the policy over options [260] or actions [52] from a single demonstration.

We evaluate NTG Networks on one-shot visual imitation learning in two domains: Block Stacking in a robot simulator [38] and Object Collection in AI2-THOR [122]. Both domains involve

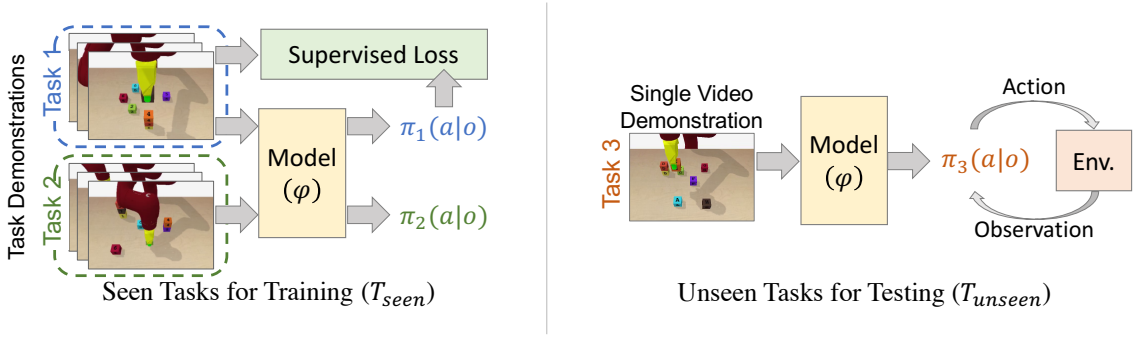


Figure 7.2: Overview of the setting of one-shot visual imitation learning. The seen tasks (Task 1 and 2) are used to train the model ϕ to instantiate the policy π_i from the demonstration. During testing, ϕ is applied to a single video demonstration from the unseen Task 3 to generate the policy π_3 to interact with the environment.

multi-step planning for interaction and are inherently compositional. We show that NTG significantly improves the data efficiency on these complex tasks for direct imitation from video by explicitly incorporating compositionality. We also show that with the data-driven task structure, NTG outperforms methods that learn unstructured task representation [52] and methods that use strong hierarchically structured supervision [260], albeit without requiring detailed supervision. Further, we evaluate NTG on real-world videos. We show that NTG can effectively predict task graph structure on the JIGSAWS [72] surgical dataset and generalize to unseen human demonstrations.

In summary, the main contributions of our work are: (1) Introducing compositionality to both the task and policy representation to enable one-shot visual imitation learning on long-horizon tasks; (2) Proposing Neural Task Graph (NTG) Networks, a novel framework that uses task graph to capture the structure and the goal of a task; (3) Addressing the challenge of novel visual state decomposition using a Conjugate Task Graph (CTG) formulation.

7.2 Related Work

Imitation Learning. Traditional imitation learning work uses physical guidance [4, 170] or teleoperation [255, 266] as demonstration. While, third-person imitation learning uses date from other agents or viewpoints [150, 217]. Recent methods for one-shot imitation learning [52, 65, 78, 258, 260, 264] can translate a single demonstration to an executable policy. The most similar to ours is NTP [260] that also learns long-horizon tasks. However, NTP (1) uses strong hierarchical frame label supervision and (2) suffers from a noticeable drop in performance with visual state. Our method

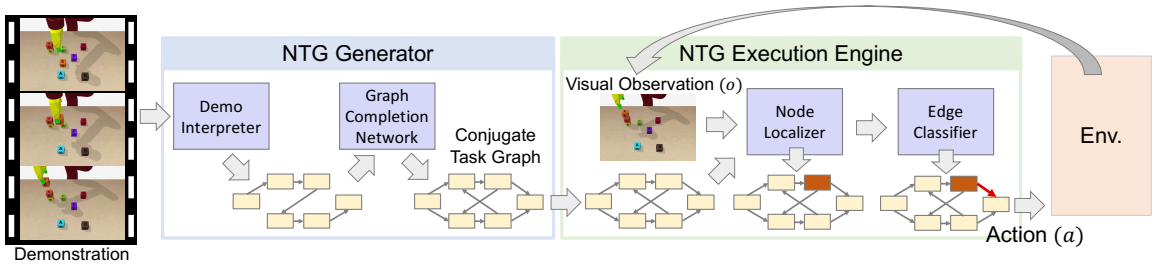


Figure 7.3: Overview of our Neural Task Graph (NTG) networks. The NTG networks consist of a generator that produces the conjugate task graph as the intermediate representation, and an execution engine that executes the graph by localizing node and deciding the edge transition in the task graph based on the current visual observation.

reduces the need for this strong supervision, requiring only the demonstration action sequence during training, while achieving a performance boost of over 25% in success rates.

Task Planning and Representations. Conventionally task planning focuses on high-level plans and low-level state spaces [62, 226]. Recent works integrate perception via deep learning [85, 187, 268]. HTN compounds low-level sub-tasks into higher-level abstraction to reduce the planning complexity [165, 208]. Other representations include: integrating task and motion planning [108] and behavior-based systems [169]. In vision, And-Or Graphs capture the hierarchical structures and have been used to parse video demonstrations [148]. Unlike previous methods, our task graph representation is data-driven and domain-agnostic: we generate nodes and edges directly from task demonstrations.

Structural Video Understanding. Generating task graphs from demonstrations is related to video understanding. Annotation in videos is hard to obtain. One solution is to use the language as supervision. This includes instructional video [5, 98, 216], movie script [234, 274], and caption annotation [83, 125]. We focus on how the structure is helpful for task learning, and assume the annotation for the seen tasks.

Compositional Models in Vision and Robotics. Recent works have utilized compositionality to improve models' generalization, including visual question answering [8, 97, 106] and policy learning [7]. We show the same principle can significantly improve data efficiency in imitation learning to enable visual learning of complex tasks.

7.3 Problem Formulation

Our goal is to learn to execute a previously unseen task from a single video demonstration. We refer to this as one-shot *visual* imitation, where the model directly learns from visual inputs. Let $\mathbb{T}$ be the set of all tasks in the domain of interest, $\mathbb{A}$ be the set of high-level actions, and $\mathbb{O}$ be the space of visual observation. A video demonstration d for a task τ is defined as a video, $d^{\tau} = [o_1, \ldots, o_T]$, that complete the task. As shown in Figure 7.2, $\mathbb{T}$ is split into two sets: $\mathbb{T}_{seen}$ with a large amount of demonstrations and supervision for training, and $\mathbb{T}_{unseen}$ with only task demonstrations for evaluation. The goal is to learn a model $\phi(\cdot)$ from $\mathbb{T}_{seen}$ that can instantiate a policy $\pi_d(a|o)$ from d to perform the tasks in $\mathbb{T}_{unseen}$ using visual observations.

The learning problem is formulated as learning a model $\phi(\cdot)$ that maps demonstration d to the policy $\phi(d) = \pi_d(a|o)$. $\mathbb{T}_{seen}$ is used to train this model with demonstrations and potentially extra supervision. At test time, given a demonstration d from an unseen task, the hope is that $\phi(\cdot)$ trained on $\mathbb{T}_{seen}$ can generalize to novel task instances in $\mathbb{T}_{unseen}$ and produce a policy that can complete the novel task illustrated by the visual demonstration.

7.4 Model

We have formulated one-shot visual imitation as learning the model $\phi(\cdot)$ that maps a video demonstration to the policy. As shown in Figure 7.1, our key contribution is explicitly incorporating *compositionality* to improve the data efficiency of generalization. We decompose $\phi(\cdot)$ into two components: a *graph generator* $\phi_{gen}(\cdot)$ that generates the task graph G from the demonstration ($G = \phi_{gen}(d)$), and a *graph execution engine* $\phi_{exe}(\cdot)$ that executes the task graph and acts as the policy ($\pi_d = \phi_{exe}(G)$). The structure of the task graph G modularizes both the demonstration and the policy. This leads to stronger data efficiency of generalizing to unseen tasks. An overview is shown in Figure 7.3.

7.4.1 Neural Task Graph Generator

The NTG Generator generates a task graph capturing the structure of an unseen task from a single video demonstration. This is challenging since the video demonstration of an unseen task introduces novel visual states that are not observed in the seen tasks. This challenge is amplified by our goal of learning from *visual* observation, which prevents direct state space decomposition. In this case, generating the traditional task graph is ill-posed due to the exploding number of nodes. We address

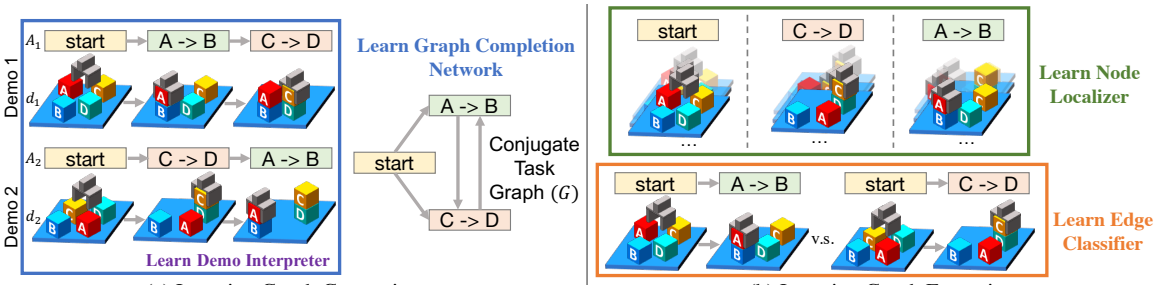
this by leveraging the conjugate relationship between state and action and work with the conjugate task graph [89], where the nodes are the actions, and the edges implicitly depend on the current state. In the experiments, we show that this scheme significantly simplifies the (conjugate) task graph generation problem.

Conjugate Task Graph (CTG). A *task graph* $\bar{G} = \{\bar{V}, \bar{E}\}$ contains nodes $\bar{V}$ as the states and $\bar{E}$ the directed edges for the transitions or actions between them. A successful execution of the task is equivalent to a path in the graph that reaches the goal node. The task graph captures the structure of the task, and the effect of each action. However, generating this graph for an unseen task is extremely challenging, as each unseen state would be mapped to a new node. This is especially the case in visual tasks, where the state space is high dimensional. We thus work with the *conjugate task graph*(CTG) [89], $G = \{V, E\}$, where the actions are now the nodes V, and the states become edges E, which implicitly encode the preconditions of the actions. This allows us to bypass explicit state modeling, while still being able to perform tasks by traversing the conjugate task graph.

We assume that all actions are observed during training from the seen tasks, which is reasonable for tasks in the same domain. This gives all the nodes in CTG, and the goal is to infer the correct edges. This can be viewed as understanding the preconditions for each action. We propose two steps for generating the edges: (i) *Demo Interpretation*: First we obtain a valid path traversing the conjugate task graph by observing the action order in the demonstration; (ii) *Graph Completion*: The second step is to add the edges that are not observed in the demonstration. There might be actions whose order can be permutated without affecting the final outcome. As we only have a single demonstration, this interchangeability is not captured in the previous step. We learn a Graph Completion Network, which adds more edges that are proper given the edges initialized by step (i).

Demo Interpreter. Given $d = [o_1, \ldots, o_T]$, our goal is to output $A = [a_1, \ldots, a_K]$, the sequence of the actions executed in the demonstration as the initial edges in the CTG as shown in Figure 7.4. The visual observations o_t are first encoded by a CNN as $Enc(o_t)$. We then adapt a seq2seq model [151] as our demo interpreter to take $Enc(o_t)$ as inputs and generate A. We do not use a frame-based classifier, as we do not need accurate per-frame action classification. What is critical here is that the sequence of actions A provides reasonable initial action order constraints (edges) to our conjugate task graph. We do assume the training demonstrations in $\mathbb{T}_{seen}$ come with the action sequence A as supervision for our demo interpreter. We only require this "flat" supervision for $\mathbb{T}_{seen}$, as opposed to the strong hierarchical supervision used in the previous work [260].

Graph Completion Network (GCN). Given a valid path (action sequence) from the demo interpreter, the goal is to complete the edges that are not observed in the demo. We formulate this as



(a) Learning Graph Generation (b) Learning Graph Execution

Figure 7.4: Illustration of our learning setting with a block stacking task. The video demonstrations d_i in the seen tasks only require corresponding action sequence A_i. We aggregate data from all the demonstrations in the same task and use it as the supervision of each component of our model. This approach allows us to bypass the need for strong supervision as in previous works.

learning graph state transitions [105, 118]. Our GCN iterates between two steps: (i) edge update and (ii) propagation. Given the node embedding $NE_{gcn}(n_i)$ for each node n_i, the edge strengths are updated as:

$$\mathscr{C}_{ij}^{t+1} = (1 - \mathscr{C}_{ij}^{t}) \cdot f_{set}(N_i^t, N_j^t) + \mathscr{C}_{ij}^{t} \cdot f_{reset}(N_i^t, N_j^t), \tag{7.1}$$

where $\mathscr{C}_{ij}^{t}$ is the adjacency matrix of the previous iteration, f_{set} and f_{reset} are MLPs for setting and resetting the edge, and $N_i = NE_{gcn}(n_i)$ is the node embedding for node i. Given $\mathscr{C}^t$ and the current node embeddings N^t, the propagation step updates the node embeddings with:

$$N_i^{t+1} = rnn(a_i, N_i^t), a_i = \sum_j \mathscr{C}_{ij}^t f_f(N_j^t) + \mathscr{C}_{ji}^t f_b(N_j^t), \tag{7.2}$$

where $rnn(a_i, N_i^t)$ takes the message a_i from other nodes as input and updates the hidden state N_i^t to N_i^{t+1}.

7.4.2 Neural Task Graph Execution

We have discussed how the NTG generates a CTG as the compositional representation of a task demonstration. Next we show how to instantiate a policy from this task graph. We propose the NTG *execution engine* that interacts with the environment by executing the task graph. The execution engine executes a task graph in two steps: (i) Node Localization: The execution engine first localizes the current node in the graph based on the visual observation. (ii) Edge Classification: For a given node, there can be multiple outgoing edges for transitions to different actions. The edge classifier checks the (latent) preconditions of each possible next action and picks the most fitting one. These two steps enable the execution engine to use the generated Conjugate Task Graph as a reactive policy which completes the task given observations. Formally, we decompose this policy as: $\pi(a|o) \propto$

$\varepsilon(a|n,o)\ell(n|o)$, where the localizer $\ell(n|o)$ localizes the current node n based on visual observation o, and the edge classifier $\varepsilon(a|n,o)$ classifies which edge transition from n and o. Deciding the edge transition given the node is equivalent to selecting the next action a.

Node Localizer. We define the localizer as: $\ell(n|o) \propto Enc(o)^T NE_{loc}(n)$, where the probability of a node is proportional to the inner product between $Enc(o)$, the encoded visual observation, and $NE_{loc}(n)$, the node embedding of the node. Since our nodes are actions that are already observed in the seen tasks, we can learn the node embeddings effectively. This shows the benefit of modularizing our policy, where sub-modules are more generalizable.

Edge Classifier. The edge classifier is the key for NTG to generalize to unseen tasks. Unlike the localizer, which is approx. invariant across seen and unseen tasks, deciding the correct edge requires the edge classifier to correctly infer the underlying states from the visual observations. Take block stacking as an example. For a task that aims to stack blocks A, B, and C in order, the robot should not pick-and-place C unless B is already *on A*. The edge classifier thus needs to recognize such prerequisites for actions involving block C.

$$\varepsilon(a|n,o) \propto (W_\varepsilon[Enc(o), NE_{gcn}(n)])^T NE_{loc}(n_a), \tag{7.3}$$

where n_a is the node for action a, and $NE_{gcn}(\cdot)$ is the final node embedding from our GCN in Section 7.4.1. As the GCN node embedding is used to generate edges in the conjugate task graph, it captures the task structure. We use NE_{loc} from localization for the destination node.

7.4.3 Learning NTG Networks

We have described how we decompose $\phi(\cdot)$ into the generator and the execution engine. As discussed in Section 7.3 we train both on $\mathbb{T}_{seen}$. In contrast to previous works that require strong supervision on $\mathbb{T}_{seen}$ (state-action pairs [52] or hierarchical supervision [260]), NTG only requires the raw visual observation along with the flat action sequence (lowest level program in [260] without a manually defined action hierarchy). An overview of learning different components of NTG is shown in Figure 7.4.

Learning Graph Generation. For each demonstration d_i^τ of task τ, we have the corresponding $A_i^\tau = [a_1, \ldots, a_K]$, the executed actions. First, we translate A_i to a path $\{P_i^\tau = (\tilde{V}, \tilde{E}_i^\tau)\}$ by using all actions as nodes $\tilde{V}$ and adding edges of the transitions in A_i to $\tilde{E}_i$. For a single task τ, we use the union of all demonstrated paths of τ as the edges $E_t = \bigcup_i \tilde{E}_i^\tau$ of the groud truth conjugate task graph $g_\tau = (V, E_t)$. In this case, the goal of GCN is to transform each P_i^τ to g_τ by completing the missing

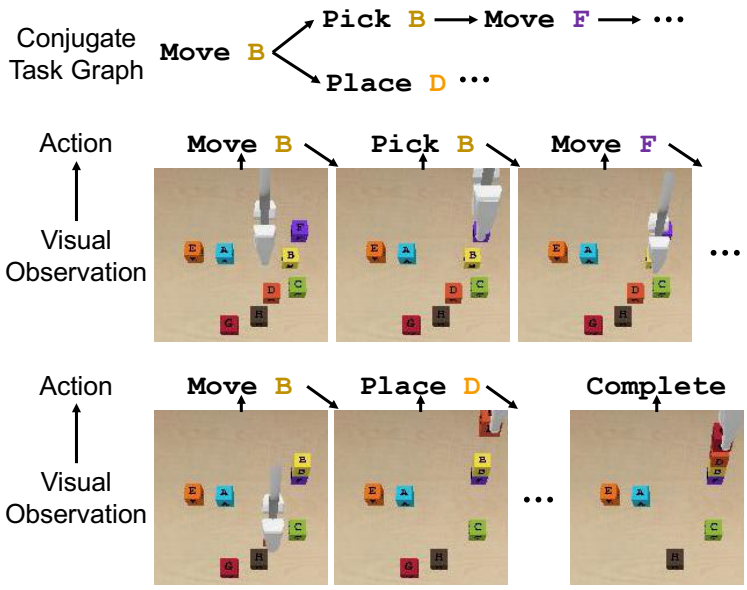


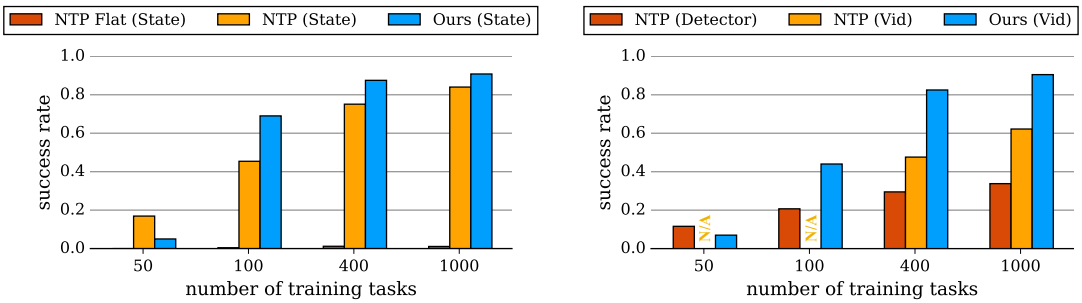
Figure 7.5: Execution of NTG based on the conjugate task graph. Although the execution engine visited the `(Move B)` node twice, it is able to correctly decide the next action using the edge classifier by understanding the second visit needs to `(Place D)`.

edges in P_i^τ. We use the binary cross entropy loss following [105] to train the GCN, where the input is P_i^τ and the goal is to generate g_τ.

Learning Graph Execution. Given a task graph from the generator, we learn an execution engine that derives the policy. As discussed in Section 7.4.2, we decompose the policy into node localizer and edge classifier. For the localizer, we use the video frames as input and the corresponding action labels from the demonstrations as targets. For the edge classifier, we collect all pairs of source-target nodes connected by transitions, and use the action label from the demonstration as the target. Additionally, the edge classifier uses the node embedding from our Graph Completion Network. The idea is that the embedding from the GCN can inform the edge classifier about what kind of visual state it should classify and learn to generalize to the unseen task.

7.5 Experiments

Our experiments aim to answer the following questions: (1) With a single *video* demonstration, how does NTG generalize to unseen tasks and compare to baselines without using compositionality? (2) How do each of the components of NTG contributes to its performance? (3) Is NTG applicable to real-world data? For the first two questions, we evaluate and perform ablation study of NTG in two



(a) Block Stacking Results with Full State

(b) Block Stacking Results with Visual State

Figure 7.6: Results for generalizing block stacking to unseen target configuration. (a) Results with the block locations as input, and (b) Results with raw video as input. Our NTG model significantly outperforms the baselines despite using only flat supervision.

challenging task domains: the Block Stacking [260] using the BulletPhysics [38] and the Object Collection task in the AI2-THOR [271]. For the last question, we evaluate NTG on real-world surgical data and examine its graph prediction and evaluation of unseen tasks on the JIGSAWS [72] dataset.

7.5.1 Evaluating Block Stacking in BulletPhysics

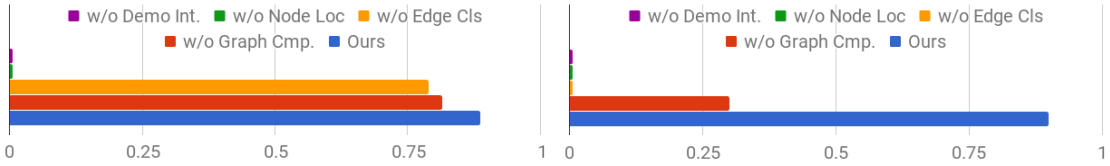
We evaluate NTG's generalization to unseen target configurations. The hierarchical structure of block stacking provides a large number of unique tasks and is ideal for analyzing the effect of explicitly introducing compositionality.

Experimental Setup. The goal of Block Stacking is to stack the blocks into a target configuration. We follow the setup in Xu *et al.* [260]. We use eight 5 cm cubes with different colors and lettered IDs. A task is considered successful if the end configuration matches the task demonstration. We use the 2000 distinct Block Stacking tasks and follow the training/testing split of Xu *et al.* [260].

Baselines. We compare to the following models:

- *Neural Task Programming (NTP)* [260] learns to synthesize policy from demonstration by decomposing a demonstration recursively. In contrast to ours, NTP assumes strong structural supervision: both the program hierarchy and the demonstration decomposition are required at training. We use NTP as an example of methods that encourage compositionality via strong structural supervision.

- *NTP Flat* is an ablation of NTP, which only uses the same supervision as our NTG model



(a) Block Stacking Ablation Study (b) Object Sorting Ablation Study

Figure 7.7: Ablation study of NTG. (a) Demo Int. and Node Loc. are almost indispensable. (b) Both GCN and Edge Cls are required to generalize to execution order different from the demonstration.

(lowest level program).

- *NTP (Detector)* first detects the block and feeds that into the model as the approximated full state. The detector is trained separately with additional supervision.

Results. Results are shown in Figure 7.6. The x-axis is the number of training seen tasks. We compare models with full state (State) and visual state (Vid) as input. Full state uses the 3D block location, and the visual state uses 64×64 RGB frames. For both input modalities, NTG can capture the structure of the tasks and generalize better to unseen target configuration compared to the baseline. NTG with raw visual input (Ours (Vid)) performs on-par with NTP using full state (NTP (State)). When there is not enough training data (50 tasks), the NTP (State) and NTP (Detector) in are able to outperform NTG because of the extra supervision (hierarchical for NTP (State), and detection for NTP (Detector)). However, once NTG is trained with more than 100 tasks, it is able to quickly interpret novel tasks and significantly outperforms the baselines. Figure 7.5 shows an NTG execution trace. Although the execution engine visited the `(Move B)` node twice, it is able to correctly decide the next action based on the visual observation by interpreting the underlying state from the visual observation.

7.5.2 Ablation Analysis of NTG Model Components

Before evaluating other environments, we analyze the importance of each component of our model. Some sub-systems are almost *indispensable*. For example, without the Demo Interpreter, there is no information from the video demonstration, and the policy is no longer task-conditional. We perform the ablation study using 1000 training tasks as follows: For Demo Interpreter, we initialize CTG as a fully connected graph without order constraints from the demonstration. For Node Localizer and Edge Classifier, we replace the corresponding term in the policy $\pi(a|o) \propto \varepsilon(a|n,o)\ell(n|o)$ by a constant. For GCN, we skip the graph completion step. As shown in Figure 7.7(a), the policy

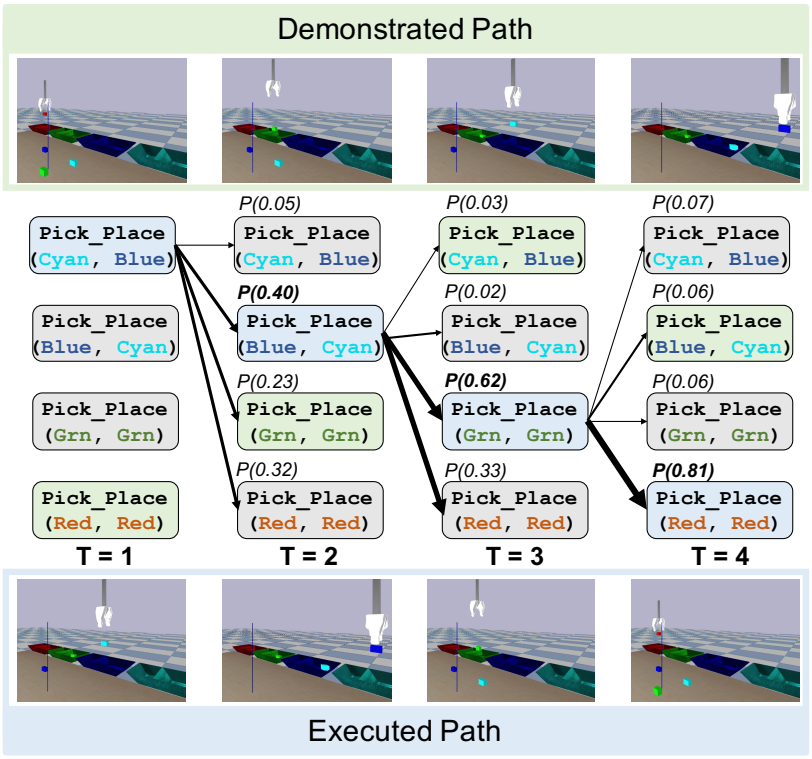
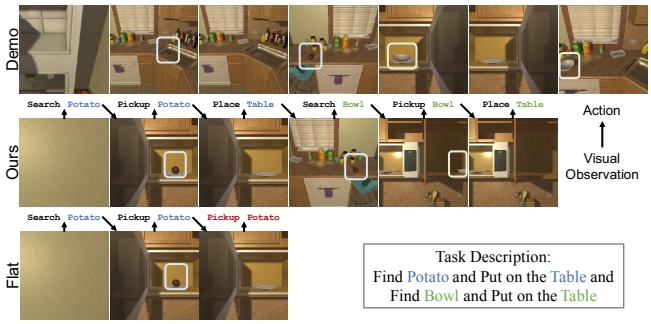


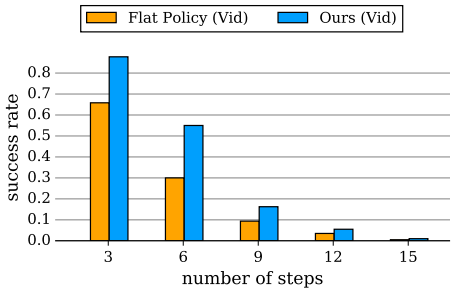
Figure 7.8: Using GCN, our policy is able to solve an unseen sorting task in a different order than the provided demonstration.

cannot complete any of the tasks without Demo Interpreter or Node Localizer. While our full model still performs the best, removing Edge Classifier or GCN does not give as big a performance gap. This is because the Block Stacking tasks from [260] do not all require task structure understanding.

Alternate Solutions for Task. GCN is particularly important for situations requiring alternative execution orders. For example, the task of "putting the red ball into the red bin and the blue ball into the blue bin". It is obvious to us that we can either put the red ball first or the blue ball first. This ability to generalize to alternative execution orders is exactly what we aim to capture with GCN. Without GCN, the policy can be easily stuck at unseen execution order (*i.e.,* not understanding object sorting order can be swapped). We thus analyze GCN on the "Object Sorting" task (details in Section VI of [260]), but initialize the scene to require execution order different from the demonstration. These settings will occur often when the policy needs to recover from failure or complete a partially completed tasks. This is challenging because: (i) GCN has to generalize and introduce alternative execution order beyond the demonstration. (ii) Edge Classifier needs to correctly select



(a) Example Task Execution Comparing Ours to Flat Policy



(b) Object Collection Results

Figure 7.9: (a) Object Collection results. The bounding boxes are only for visualization and is not used anywhere in our model. The objects can appear in locations that are different from the demonstration, which leads to challenging and diverse visual state. NTG is able to understand the underlying state (e.g., if the object is found) from the visual input and successfully complete the task. (b) Object Collection results on varying numbers of steps. The NTG model is only trained with 6 and 12 steps, and is able to generalize well to other numbers of steps.

the action from the newly introduced edges by GCN. As shown in Figure 7.7(b), the policy cannot complete any of the tasks without Edge Classifier because of the ambiguities in the completed task graph. Figure 7.8 shows a qualitative example of how our method learns to complete "Object Sorting" with order different from the demonstration using GCN. This shows the importance of both the Edge Classifier and GCN, which are required to complete this challenging task.

7.5.3 Evaluating Object Collection in AI2-THOR

In this experiment, we evaluate the Object Collection task, in which an agent collects and drop off objects from a wide range of locations with varying visual appearances. We use AI2-THOR [271] as the environment, which allows the agent to navigate and interact with objects via semantic actions (e.g., `Open`). This task is more complicated than block stacking because: First, the agent is navigating in the scene and thus can only have partial observations. Second, the photo-realistic simulation enables a variety of visual appearance composition. In order to complete the task, the model needs to understand various appearances of the object and location combinations.

Experimental Setup. An Object Collection task involves visiting M randomly selected searching locations for a set of N target objects out of C categories. Upon picking up a target object, the agent visits and drops off the object at one of K designated drop-off receptacles. A task is considered successful if all of the target objects are placed at their designated receptacles at the end of the task episode. The available semantic actions are `search`, `pickup(object)`,

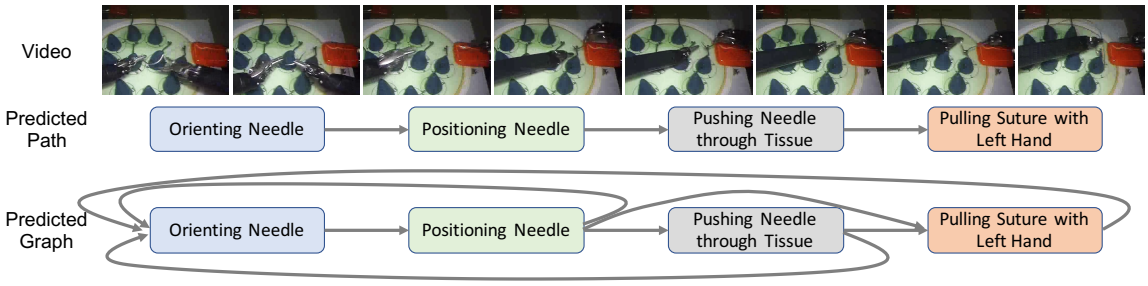


Figure 7.10: Part of a predicted graph from a single demonstration of an unseen task on the JIGSAWS dataset. Our method is able to learn that for the Needle Passing task, after failing any of the step in this subtask, the agent should restart by reorienting the needle.

`dropoff(receptacle)`. The `search` action visits each searching locations in a randomized order. `pickup(object)` picks up a selected object and the action would fail if the selected object is not visible to the agent. `dropoff(receptacle)` would teleport the agent to a selected drop-off receptacle (`tabletop`, `cabinet`, etc) and drop off. We use $N = [1,5]$ objects (3-15 steps) out of $C = 8$ categories, $M = N + 3$ search locations, and $K = 5$ drop-off receptacles.

Baseline. We compare to the "Flat Policy" baseline in [52] to show the importance of incorporating compositionality to the policy. At each step, the Flat Policy uses attention to extract relevant information from the demonstration and combine it with the observation to decide action. For a fair comparison, we implement the Flat Policy using the same architecture as our demo interpreter. Note that the Object Collection domain doesn't have hand-designed hierarchy. Hence NTP [260] is reduced to a similar flat policy model.

Results. The results for Object Collection are shown in Figure 7.9(b). The models are only trained on 2 and 4 objects and generalize to 1, 3, 5 objects. NTG significantly outperforms the Flat Policy on all numbers of objects. This shows the importance of explicitly incorporating compositionality. Qualitative comparison is shown in Figure 7.9(a). The bounding boxes are for visualization only and are not used in the model. During evaluation, the objects of interest can appear in locations that are different from the demonstration and thus lead to diverse and challenging visual appearances. It is thus important to understand the structure of the demonstration instead of naive appearance matching. Our explicit model of the task structure sets NTG apart from the flat policy and leads to stronger generalization to unseen tasks.

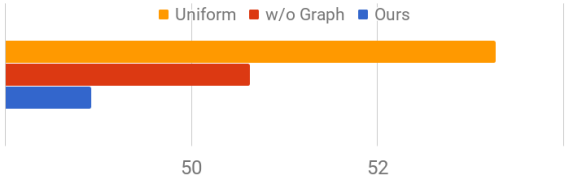


Figure 7.11: Negative log-likehood (NLL) of expert demonstrations on the JIGSAWS dataset. The policy generated by our full model can best capture the actions performed in human demonstration.

7.5.4 Evaluating Real-world Surgical Data

We have shown that NTG significantly improves one-shot visual imitation learning by explicitly incorporating compositionality. We now evaluate if this structural approach can be extended to the challenging real-world surgical data from the JIGSAWS dataset [72], which contains videos and states for surgical tasks, and the associated atomic action labeling. In this setting, our goal is to assess NTG's ability to generalize to the task of "Needle Passing", after training on the tasks of "Knot Tying" and "Suturing". This is especially challenging because it requires generalization to a new task with significant structural and visual differences, given only 2 task types for training.

Without a surgical environment, we cannot directly evaluate the policy learned by NTG on the JIGSAWS dataset. Therefore, we evaluate how well the NTG policy is able to predict what a human will do in other demonstrations. This entails generating a policy conditioned on a single demonstration of "Needle passing", and using it to evaluate the negative log-likelihood (NLL) of all the other demonstrations in the "Needle Passing" task. A lower negative log likelihood corresponds to the generated policy better explaining the other demonstrations, and in turn better capturing the task structure.

The results are shown in Figure 7.11. We compare to the no graph variant of our model and also the lower bound of a uniform policy. Unsurprisingly, the *uniform* policy performs the worst without capturing anything from the demonstration. The *no-graph* variant is able to capture some parts of the expert policy and better capture the expert demonstration. However, the policy generated by full NTG model substantially improves the NLL and is the most consistent with the expert demonstration.

In addition, we show qualitative results of part of our task graph prediction on the JIGSAWS dataset in Figure 7.10. Again, we train on "Knot Tying" and "Suturing" and evaluate on "Needle Passing". By comparing the predicted path and the final predicted graph, we can see that our model is able to introduce several new edges going back to the action "Orienting Needle". This captures

the behavior that when the execution fails in any of step in this subtask of "Needle Passing", the agent should return to "Orienting Needle" and reorient the needle to restart the subtask. This is consistent with our intuition and the ground truth graph.

7.6 Discussions

We presented Neural Task Graph (NTG) Network, a one-shot visual imitation learning method that explicitly incorporates task compositionality into both the intermediate task representation and the policy. Our novel Conjugate Task Graph (CTG) formulation effectively handles unseen visual states and serves as a reactive and executable policy. We demonstrate that NTG is able to outperform both methods with unstructured representation [52], and methods with a hand-designed hierarchical structure [260] on a diverse set of tasks, including simulated environment with photo-realistic rendering and a real-world dataset.

Chapter 8

Conclusions

8.1 Summary

In this dissertation, we have discussed our research that enables robots to interact with the unstructured world through their senses, to flexibly perform a wide range of tasks, and to adaptively learn new tasks. These works have demonstrated that the tightly coupled perception-action loop leads to more robust and generalizable robot behaviors in a variety of practical scenarios. They highlight the importance of integrating perception and action for building general-purpose robot intelligence

To recapitulate, we introduced our work that dissects the perception-action loop at three levels of abstraction (also see Table 1.1): first, learning *primitive skills* that transform raw sensory information to control commands that drive the motions of robots (Chapter 2 and 3); second, performing *sequential tasks* in interactive environments via a sequential composition of the primitive skills (Chapter 4 and 5); and third, scaling to *hierarchical tasks* that are long-horizon tasks that possess latent compositional structures (Chapter 6 and 7). We have developed novel machine learning and computer vision algorithms, combined with the domain knowledge in robotics, to address the fundamental challenges of robotic perception and control in the real world.

8.2 Future Directions

Moving forward, I am broadly interested in two parallel themes of future research centered around the perception-action loop: 1) using robots as an embodied platform to develop interactive perception, and 2) using perception as input to develop closed-loop robot control. Within these two themes, I identify the following three research topics that extend my past and ongoing research

towards tackling some of the open challenges in building general-purpose autonomy.

Building Coherent Models of the 3D World

I have explored the problem of building structured representations of images through grounded visual question answering [269, 270] and scene graph generation [126, 261]. Such representations have been proven to enhance generalization and interpretability of image understanding models. However, representations learned from static images lack sufficient 3D and temporal information crucial for robotics. Moving from my prior work, I plan to study how to build coherent models of the 3D world that capture the semantic, physical, and functional properties of objects and scenes. I believe that solving realistic tasks in robotics requires a synergy between model-based and data-driven methods. Therefore, I intend to investigate how to integrate such models of the 3D world with data-driven approaches, such as end-to-end deep learning.

Multimodal Perception Beyond Vision

The majority of my prior work has focused on visual perception as the primary sensory modality. However, humans make contact with the physical world through a vast array of sensory modalities. I envision that an intelligent robot should eventually interact with the environments through a kaleidoscope of multimodal senses, including vision, touch, sound, etc. Our work on multimodal representation learning, detailed in Chapter 3, has shed light on the merit of combining vision and force sensory modalities in contact-rich manipulation. Our experience of building the RoboTurk system [155] has also showcased the value of providing simultaneous visual and haptic feedback in remote robot teleoperation. Inspired by these two lines of work, I intend to explore multimodal perception in the following two directions: 1) learning compact representations from multimodal sensors for robot control, and 2) using multimodal feedback to design intuitive control interfaces for robot teleoperation.

Knowledge-based Reasoning for Robotics

Today's robot learning algorithms have mostly focused on specializing in short-horizon tasks in narrow domains. A general-purpose robot should be able to constantly learn and adapt through its prolonged interaction with the world. I argue that one of the key missing pieces in the current robot learning paradigm is a never-ending learning mechanism for robots to incrementally acquire new knowledge from its experiences. My past work on knowledge-based reasoning [267, 273]

has focused on building large-scale knowledge graphs for visual reasoning. To build a knowledge-based reasoning framework for general- purpose robots, I plan to study two open questions: 1) how to devise a principled framework to represent and reason about heterogeneous types of knowledge, both declarative (e.g., object detectors) and procedural (e.g., motor skills), and 2) how to continually harvest new knowledge and incorporate it into the existing repertoire.

8.3 Final Remarks

My long-term research goal is to build a computational framework for general-purpose robots to continually develop its cognitive and physical skills in real-world environments. I hope that the process of building intelligence for robots can offer new insights to understanding the developmental process of human intelligence.

I envision that robotics and automation, powered by new advances in artificial intelligence, is on the cusp of bringing about profound social and economic impacts. As a witness and practitioner in the new era of technological innovations, it is critical to align my long-term research objectives with the fundamental human values. My research enthusiasm stems from convictions for AI's positive implications for human well-being. General-purpose robots have a great potential of empowering humans in the real world, from assisting people in need and to bringing humans away from hazardous environments. Building intelligent and capable robots is not to diminish or to replace us, but to enrich us. My ultimate goal for building general-purpose robot autonomy is to augment humans' physical and intellectual capacities. As we strive to make future robot intelligence more human-like, I aspire to see that it will, in turn, make us humans more human.

Appendix A

Reinforcement and Imitation Learning

A.1 Experiment Details

The policy network takes the pixel observation and the proprioceptive feature as input. The pixel observation is an RGB image of size $64 \times 64 \times 3$. We used the Kinect for Xbox One camera[1] in the real environment. The proprioceptive feature describes the joint positions and velocities of the Kinova Jaco arm.[2] Each joint position is represented as the sin and cos of the angle of the joint in joint coordinates. Each joint velocity is represented as the scalar angular velocity. This results in a 24-dimensional proprioceptive feature that contains the positions (12-d) and velocities (6-d) of the six arm joints and the positions (6-d) of the three fingers. We exclude the finger velocities due to the noisy sensory readings on the real robot. When collecting demonstrations, we use a 6-DoF SpaceNavigator motion controller[3] to command the end effector to complete the tasks.

We used Adam [117] to train the neural network parameters. We set the learning rate of policy and value to 10^{-4} and 10^{-3} respectively, and 10^{-4} for both the discriminator and the auxiliary tasks. The pixel observation is encoded by a two-layer convolutional network. We use 2 convolutional layers followed by a fully-connected layer with 128 hidden units. The first convolutional layer has 16 8×8 filters with stride 4 and the second 32 4×4 filters with stride 2. We add a recurrent layer of 100 LSTM units before the policy and value outputs. The policy output is the mean and the standard deviation of a conditional Gaussian distribution over the 9-dimensional joint velocities. The initial policy standard deviation is set to $\exp(-3)$ for the *clearing table with blocks* task and $\exp(-1)$ for

[1] https://www.xbox.com/en-US/xbox-one/accessories/kinect
[2] http://www.kinovarobotics.com
[3] https://www.3dconnexion.com/spacemouse_compact

the other five tasks. The auxiliary head of the policy contains a separate three-layer MLP sitting on top of the convolutional network. The first two layers of the MLP has 200 and 100 hidden units respectively, while the third layer predicts the auxiliary outputs. Finally, the discriminator is a simple three-layer MLP of 100 and 64 hidden units for the first two layers with the third layer producing log probabilities. The networks use tanh nonlinearities.

We trained the visuomotor policies using the distributed PPO algorithm [92] with synchronous gradient updates from 256 CPU workers. Each worker runs the policy to complete an entire episode before the parameter updates are computed. We set a constant episode length for each task based on its difficulty, with the longest being 1000 time steps (50 seconds) for the *clearing table with blocks* and *order fulfillment* tasks. We set $K = 50$ as the number of time steps for computing K-step returns and truncated backpropagation through time to train the LSTM units. After a worker collects a batch of data points, it performs 50 parameter updates for the policy and value networks, 5 for the discriminator and 5 for the auxiliary prediction network.

A.2 Sim2Real Details

To better facilitate sim2real transfer, we lower the frequency at which we sample the observations. Pixel observations are only observed at the rate of 5Hz despite the fact that our controller runs at 20Hz. Similarly, the proprioceptive features are observed at a rate of 10Hz. In addition to observation delays, we also apply domain variations. Gaussian noise (of standard deviation 0.01) are added proprioceptive features. Uniform integers noise in the range of $[-5,5]$ are added to each pixel independently. Pixels of values outside the range of $[0,255]$ are clipped. We also vary randomly the shade of grey on the Jaco arm, the color of the table top, as well as the location and orientation of the light source (see Figure A.1).

In the case of block lifting, we vary in addition the dynamics of the arm. Specifically, we dynamically change the friction, damping, armature, and gain parameters of the robot arm in simulation to further enhance the agent's robustness.

Action Dropping. Our analysis indicates that, on the real robot, there is often a delay in the execution of actions. The amount of delay also varies significantly. This has an adverse effect on the performance of our agent on the physical robot since our agents' performance depends on the timely execution of their actions. To better facilitate the transfer to the real robot, we fine-tune our trained agent in simulation while subjecting them to a random chance of dropping actions. Specifically, each action emitted by the agent has a 50% chance of being executed immediately in which case the

Figure A.1: Tiles show the representative range of diversity seen in the domain-randomized variations of the colors, lighting, background, etc.

action is flagged as the last executed action. If the current action is not executed, the last executed action will then be executed. Using the above procedure, we fine-tune our agents on both block lifting and block stacking for a further 2 million iterations.

To demonstrate the effectiveness of action dropping, we compare our agent on the real robot over the task of block lifting. Without action dropping, the baseline agent lifts 48% percent of the time. After fine-tuning using action dropping, our agent succeeded 64% percent of the time. For the complete set of results, please see Table A.1 and Table A.2.

A.3 Task Details

We use a fixed episode length for each task, which is determined by the amount of time a skilled human demonstrator can complete the task. An episode terminates when a maximum number of agent steps are performed. The robot arm operates at a control frequency of 20Hz, which means each time step takes 0.05 second.

We segment into a sequence of stages that represent an agent's progress in a task. For instance, the *block stacking* task can be characterized by three stages, including *reaching the block*, *lifting the block* and *stacking the block*. We define functions on the underlying physical state to determine the stage of a state. This way, we can cluster demonstration states according to their corresponding stages. These clusters are used to reset training episodes in our demonstration as a curriculum technique proposed in Section 2.3.2. The definition of stages also gives rise to a convenient way of specifying the reward functions without hand-engineering a shaping reward. We define a piecewise

Table A.1: Block lifting success rate from different positions (LL, LR, UL, UR, and C represent the positions of lower left, lower right, upper left, upper right, and center respectively).

	LL	LR	UL	UR	C	All
No Action Dropping	2/5	2/5	1/5	3/5	4/5	12/25
Action Dropping	4/5	4/5	4/5	0/5	4/5	16/25

Table A.2: Success rate of the block stacking agent (with action dropping) from different starting positions (Left and Right indicate the positions of the support block upon initialization).

	Left	Right	All
Stacking Success Rate	5/10	2/10	7/20
Lifting Success Rate	9/10	7/10	16/20

constant reward function for each task, where we assign the same constant reward to all the states that belong to the same stage. We detail the stages, reward functions, auxiliary tasks, and object-centric features for the six tasks in our experiments.

Block lifting. Each episode lasts 100 time steps. We define three stages and their rewards (in parentheses) to be initial (0), reaching the block (0.125) and lifting the block (1.0). The auxiliary task is to predict the 3D coordinates of the color block. The object-centric feature consists of the relative position between the gripper and the block.

Block stacking. Each episode lasts 500 time steps. We define four stages and their rewards to be initial (0), reaching the orange block (0.125), lifting the orange block (0.25), and stacking the orange block onto the pink block (1.0). The auxiliary task is to predict the 3D coordinates of the two blocks. The object-centric feature consists of the relative positions between the gripper and the two blocks respectively.

Clearing table with blocks. Each episode lasts 1000 time steps. We define five stages and their rewards to be initial (0), reaching the orange block (0.125), lifting the orange block (0.25), stacking the orange block onto the pink block (1.0), and lifting both blocks off the ground (2.0). The auxiliary task is to predict the 3D coordinates of the two blocks. The object-centric feature consists of the 3D positions of the two blocks as well as the relative positions between the gripper and the two blocks respectively.

Clearing table with a box. Each episode lasts 500 time steps. We define five stages and their rewards to be initial (0), reaching the toy (0.125), grasping the toy (0.25), putting the toy into the

box (1.0), and lifting the box (2.0). The auxiliary task is to predict the 3D coordinates of the toy and the box. The object-centric feature consists of the 3D positions of the toy and the box as well as the relative positions between the gripper and these two objects respectively.

Pouring liquid. Each episode lasts 500 time steps. We define three stages and their rewards to be initial (0), grasping the mug (0.05), pouring ($0.1N$), where N is the number of small spheres in the other container. The auxiliary task is to predict the 3D coordinates of the mug. The object-centric feature consists of the 3D positions of the mug, the relative position between the gripper and the mug, and the relative position between the mug and the container.

Order fulfillment. Each episode lasts 1000 time steps. The number of objects varies from 1 to 4 across episodes. We define five stages that correspond to the number of toys in the boxes. The immediate reward corresponds to the number of toys placed in the correct boxes (number of toy planes in the green box and toy cars in the red box). To handle the variable number of objects, we only represent the objects nearest to the gripper for the auxiliary task and the object-centric feature. The auxiliary task is to predict the 3D coordinates of the nearest plane and the nearest car to the gripper. The object-centric feature consists of the relative positions from the gripper to these two nearest objects.

Bibliography

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Fares Abu-Dakka, Bojan Nemec, Jimmy Jørgensen, Thiusius Savarimuthu, Norbert Krüger, and Aleš Ude. Adaptation of manipulation skills in physical contact with the environment to reference force profiles. *Autonomous Robots*, 39(2):199–217, 2015.

[3] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems*, pages 5074–5082, 2016.

[4] Baris Akgun, Maya Cakmak, Karl Jiang, and Andrea Thomaz. Keyframe-based learning from demonstration. *International Journal of Social Robotics*, 4(4):343–355, 2012.

[5] Jean-Baptiste Alayrac, Piotr Bojanowski, Nishant Agrawal, Josef Sivic, Ivan Laptev, and Simon Lacoste-Julien. Unsupervised learning from narrated instruction videos. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[6] Michael Anderson. Embodied cognition: A field guide. *Artificial intelligence*, 2003.

[7] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, 2017.

[8] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[9] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[10] Brenna Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[11] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H Campbell, and Sergey Levine. Stochastic variational video prediction. *arXiv preprint arXiv:1710.11252*, 2017.

[12] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *AAAI Conference on Artificial Intelligence*, 2017.

[13] André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *Advances in Neural Information Processing Systems*, 2017.

[14] André Barreto, Doina Precup, and Joelle Pineau. Practical kernel-based reinforcement learning. *Journal of Machine Learning Research*, 2016.

[15] Yasemin Bekiroglu, Renaud Detry, and Danica Kragic. Learning tactile characterizations of object- and pose-specific grasps. In *International Conference on Intelligent Robots and Systems*, pages 1554–1560, 2011.

[16] Yasemin Bekiroglu, Dan Song, Lu Wang, and Danica Kragic. A probabilistic framework for task-oriented grasp stability assessment. In *IEEE International Conference on Robotics and Automation*, pages 3040–3047, 2013.

[17] Marc Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[18] A Bicchi, M Bergamasco, P Dario, and A Fiorillo. Integrated tactile sensing for gripper fingers. In *International Conference on Robot Vision and Sensory Control*, 1988.

[19] Aude Billard, Sylvain Calinon, Ruediger Dillmann, and Stefan Schaal. Robot programming by demonstration. In *Handbook of Robotics*, 2008.

[20] Randolph Blake, Kenith Sobel, and Thomas James. Neural synergy between kinetic vision and touch. *Psychological Science*, 15(6):397–402, 2004.

[21] Jeannette Bohg, Karol Hausman, Bharath Sankaran, Oliver Brock, Danica Kragic, Stefan Schaal, and Gaurav Sukhatme. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33:1273–1291, December 2017.

[22] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 2008.

[23] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots. *IEEE Transactions on Systems, Man and Cybernetics*, 1989.

[24] Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 1991.

[25] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 182–189, 2011.

[26] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *International Conference on Robotics and Automation*, pages 4243–4250, 2018.

[27] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal on Robotics and Automation*, 1986.

[28] Jonathon Cai, Richard Shin, and Dawn Song. Making neural programming architectures generalize via recursion. *International Conference on Learning Representations*, 2017.

[29] Michael Caine, Tomás Lozano-Pérez, and Warren P Seering. Assembly strategies for chamferless parts. In *International Conference on Robotics and Automation*, 1989.

[30] Roberto Calandra, Andrew Owens, Dinesh Jayaraman, Justin Lin, Wenzhen Yuan, Jitendra Malik, Edward Adelson, and Sergey Levine. More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters*, 3(4):3300–3307, 2018.

[31] Roberto Calandra, Andrew Owens, Manu Upadhyaya, Wenzhen Yuan, Justin Lin, Edward Adelson, and Sergey Levine. The feeling of success: Does touch sensing help predict grasp outcomes? *Conference on Robot Learning*, 2017.

[32] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *International Conference on Robotics and Automation*, 2017.

[33] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *International Conference on Computer Vision*, pages 2722–2730, 2015.

[34] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Conference on Computer Vision and Pattern Recognition*, 2005.

[35] Michael Jae-Yoon Chung, Andrzej Pronobis, Maya Cakmak, Dieter Fox, and Rajesh Rao. Autonomous question answering with mobile robots in human-populated environments. In *International Conference on Intelligent Robots and Systems*, 2016.

[36] F. Conti, F. Barbagli, R. Balaniuk, M. Halg, C. Lu, D. Morris, L. Sentis, J. Warren, O. Khatib, and K. Salisbury. The CHAI libraries. In *Eurohaptics*, pages 496–500, 2003.

[37] François Conti and Oussama Khatib. A framework for real-time multi-contact multi-body dynamic simulation. In *Robotics Research*, pages 271–287, 2016.

[38] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation, games, robotics and machine learning. `http://pybullet.org/`, 2016–2017.

[39] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75(3):297–325, 2009.

[40] Andrew Davison. Real-time simultaneous localisation and mapping with a single camera. In *International Conference on Computer Vision*, 2003.

[41] Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 1993.

[42] Feras Dayoub, Timothy Morris, Ben Upcroft, and Peter Corke. Vision-only autonomous navigation using topometric maps. In *International Conference on Intelligent Robots and Systems*, 2013.

[43] Avik De, Karl Bayer, and Daniel Koditschek. Active sensing for dynamic, non-holonomic, robust visual servoing. In *International Conference on Robotics and Automation*, 2014.

[44] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, 2018.

[45] Marc Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2013.

[46] Marc Deisenroth and Carl Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*, 2011.

[47] Misha Denil, Pulkit Agrawal, Tejas Kulkarni, Tom Erez, Peter Battaglia, and Nando de Freitas. Learning to perform physics experiments via deep reinforcement learning. *arXiv preprint arXiv:1611.01843*, 2016.

[48] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer. *International Conference on Robotics and Automation*, 2017.

[49] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel rahman Mohamed, and Pushmeet Kohli. RobustFill: Neural Program Learning under Noisy I/O. *International Conference on Machine Learning*, 2017.

[50] Christian Dornhege, Marc Gissler, Matthias Teschner, and Bernhard Nebel. Integrating symbolic and geometric planning for mobile manipulation. In *International Workshop on Safety, Security & Rescue Robotics*, 2009.

[51] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations*, 2017.

[52] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in Neural Information Processing Systems*, 2017.

[53] Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas Griffiths, and Alexei Efros. Investigating human priors for playing video games. *International Conference on Machine Learning*, 2018.

[54] Gerald Edelman. *Neural Darwinism: The theory of neuronal group selection*. Basic Books, 1987.

[55] Clemens Eppner, Raphael Deimel, José Alvarez-Ruiz, Marianne Maertens, and Oliver Brock. Exploitation of environmental constraints in human and robotic grasping. *International Journal of Robotics Research*, 34(7):1021–1038, June 2015.

[56] Linxi Fan*, Yuke Zhu*, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. *Conference on Robot Learning*, 2018.

[57] Kuan Fang, Yuke Zhu, Animesh Garg, Virja Mehta, Andrey Kuryenkov, Li Fei-Fei, and Silvio Savarese. Learning task-oriented grasping for tool manipulation with simulated self-supervision. *Robotics: Science and Systems*, 2018.

[58] Alireza Fathi and James Rehg. Modeling actions through state changes. In *Conference on Computer Vision and Pattern Recognition*, 2013.

[59] Nima Fazeli, Samuel Zapolsky, Evan Drumwright, and Alberto Rodriguez. Fundamental limitations in performance and interpretability of common planar rigid-body contact models. *arXiv preprint arXiv:1710.04979*, 2017.

[60] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[61] Richard Fikes, Peter Hart, and Nils Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 1972.

[62] Richard Fikes and Nils Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971.

[63] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *International Conference on Robotics and Automation*, pages 2786–2793, 2017.

[64] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[65] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, 2017.

[66] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[67] Roy Fox, Sanjay Krishnan, Ion Stoica, and Ken Goldberg. Multi-Level Discovery of Deep Options. In *arXiv preprint arXiv:1703.08294*, 2017.

[68] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *International Conference on Intelligent Robots and Systems*, pages 4019–4026, 2016.

[69] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[70] Shameek Ganguly and Oussama Khatib. Experimental studies of contact space model for multi-surface collisions in articulated rigid-body systems. In *International Symposium on Experimental Robotics*. Springer, 2018.

[71] Yang Gao, Lisa Anne Hendricks, Katherine Kuchenbecker, and Trevor Darrell. Deep learning for tactile understanding from visual and haptic data. In *International Conference on Robotics and Automation*, pages 536–543, 2016.

[72] Yixin Gao, Swaroop Vedula, Carol Reiley, Narges Ahmidi, Balakrishnan Varadarajan, Henry Lin, Lingling Tao, Luca Zappella, Benjamn Béjar, David Yuh, Chi Chiung Grace Chen, René Vidal, Sanjeev Khudanpur, and Gregory Hager. JHU-ISI Gesture and Skill Assessment Working Set (JIGSAWS): A Surgical Activity Dataset for Human Motion Modeling. In *Workshop on Modeling and Monitoring of Computer Assisted Interventions, MICCAI*, 2014.

[73] Cristina Garcia Cifuentes, Jan Issac, Manuel Wüthrich, Stefan Schaal, and Jeannette Bohg. Probabilistic articulated real-time tracking for robot manipulation. *IEEE Robotics and Automation Letters*, 2(2):577–584, April 2017.

[74] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[75] James Gibson. *The ecological approach to visual perception: classic edition*. Psychology Press, 2014.

[76] Clyde Lee Giles, Clifford Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Learning*, 4(3), 2008.

[77] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Conference on Computer Vision and Pattern Recognition*, 2014.

[78] Wonjoon Goo and Scott Niekum. One-shot learning of multi-step tasks from observation via activity localization in auxiliary video. *arXiv preprint arXiv:1806.11244*, 2018.

[79] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[80] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *International Conference on Robotics and Automation*, 2017.

[81] Shixiang Gu, Tim Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. In *International Conference on Machine Learning*, 2016.

[82] Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Advances in Neural Information Processing Systems*, 2014.

[83] Abhinav Gupta, Praveen Srinivasan, Jianbo Shi, and Larry Davis. Understanding videos, constructing plots learning a visually grounded storyline model from annotated videos. In *Conference on Computer Vision and Pattern Recognition*, 2009.

[84] Abhishek Gupta, Clemens Eppner, Sergey Levine, and Pieter Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstrations. In *International Conference on Intelligent Robots and Systems*, pages 3786–3793, 2016.

[85] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[86] Haddad Haddad, Maher Khatib, Simon Lacroix, and Raja Chatila. Reactive navigation in outdoor environments using potential fields. In *International Conference on Robotics and Automation*, 1998.

[87] Ankur Handa, Viorica Patraucean, Vijay Badrinarayanan, Simon Stent, and Roberto Cipolla. Understanding real world indoor scenes with synthetic data. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[88] Ankur Handa, Viorica Patraucean, Simon Stent, and Roberto Cipolla. SceneNet: An annotated model generator for indoor scene understanding. In *International Conference on Robotics and Automation*, 2016.

[89] Bradley Hayes and Brian Scassellati. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *International Conference on Robotics and Automation*, 2016.

[90] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *International Conference on Computer Vision*, 2017.

[91] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*, 2016.

[92] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[93] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

[94] Nicolas Heess, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2926–2934, 2015.

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

[96] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[97] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *International Conference on Computer Vision*, 2017.

[98] De-An Huang, Shyamal Buch, Lucio Dery, Animesh Garg, Li Fei-Fei, and Juan Carlos Niebles. Finding "it": Weakly-supervised reference-aware visual grounding in instructional videos. In *Conference on Computer Vision and Pattern Recognition*, 2018.

[99] De-An Huang*, Suraj Nair*, Danfei Xu*, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural Task Graphs: Generalizing to Unseen Tasks from a Single Video Demonstration. *Conference on Computer Vision and Pattern Recognition*, 2019.

[100] Minyoung Huh, Pulkit Agrawal, and Alexei Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

[101] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural Computation*, 25(2):328–373, 2013.

[102] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[103] Stephen James, Andrew Davison, and Edward Johns. Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *arXiv preprint arXiv:1707.02267*, 2017.

[104] Todd Jochem, Dean Pomerleau, and Charles Thorpe. MANIAC: A next generation neurally based autonomous road follower. In *International Conference on Intelligent Autonomous Systems*, pages 15–18, 1993.

[105] Daniel Johnson. Learning graphical state transitions. In *International Conference on Learning Representations*, 2017.

[106] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Judy Hoffman, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Inferring and executing programs for visual reasoning. In *International Conference on Computer Vision*, 2017.

[107] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo platform for artificial intelligence experimentation. In *International Joint Conferences on Artificial Intelligence*, 2016.

[108] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *International Conference on Robotics and Automation*, 2011.

[109] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

[110] Mrinal Kalakrishnan, Ludovic Righetti, Peter Pastor, and Stefan Schaal. Learning force control policies for compliant manipulation. In *International Conference on Intelligent Robots and Systems*, 2011.

[111] Daniel Kappler, Peter Pastor, Mrinal Kalakrishnan, Manuel Wuthrich, and Stefan Schaal. Data-driven online decision making for autonomous manipulation. In *Robotics: Science and Systems*, 2015.

[112] Micha Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jakowski. ViZDoom: A doom-based AI research platform for visual reinforcement learning. In *Conference on Computational Intelligence and Games*, 2016.

[113] Oussama Khatib. Inertial Properties in Robotic Manipulation: An Object-Level Framework. *International Journal of Robotics Research*, 14(1):19–36, 1995.

[114] Kiyosumi Kidono, Jun Miura, and Yoshiaki Shirai. Autonomous visual navigation of a mobile robot using a human guided experience. *Robotics and Autonomous Systems*, 2002.

[115] Dongsung Kim and Ramakant Nevatia. Symbolic navigation with a generic map. In *IEEE Workshop on Vision for Robots*, 1995.

[116] H. Jin Kim, Michael Jordan, Shankar Sastry, and Andrew Ng. Autonomous helicopter flight via reinforcement learning. In *Advances in Neural Information Processing Systems*, 2004.

[117] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[118] Thomas Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[119] Jens Kober, Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[120] Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *International Conference on Robotics and Automation*, 2004.

[121] Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *International Journal of Robotics Research*, 2008.

[122] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

[123] Kurt Konolige, James Bowman, J. D. Chen, Patrick Mihelich, Michael Calonder, Vincent Lepetit, and Pascal Fua. View-based maps. *International Journal of Robotics Research*, 2010.

[124] Satwik Kottur, José Moura, Devi Parikh, Dhruv Batra, and Marcus Rohrbach. Visual coreference resolution in visual dialog using neural module networks. In *European Conference on Computer Vision*, 2018.

[125] Ranjay Krishna, Kenji Hata, Frederic Ren, Li Fei-Fei, and Juan Carlos Niebles. Dense-captioning events in videos. In *International Conference on Computer Vision*, 2017.

[126] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalanditis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 2017.

[127] Sanjay Krishnan*, Animesh Garg*, Sachin Patil, Colin Lea, Gregory Hager, Pieter Abbeel, and Ken Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *International Journal of Robotics Research*, 2018.

[128] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[129] Tejas Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016.

[130] Tejas Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

[131] Vikash Kumar, Abhishek Gupta, Emanuel Todorov, and Sergey Levine. Learning dexterous manipulation policies from experience and imitation. *arXiv preprint arXiv:1611.05095*, 2016.

[132] Simon Lacey and Krish Sathian. Crossmodal and multisensory interactions between vision and touch. In *Scholarpedia of Touch*, pages 301–315. Springer, 2016.

[133] Brenden Lake, Tomer Ullman, Joshua Tenenbaum, and Samuel Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 2016.

[134] Tian Lan*, Yuke Zhu*, Amir Roshan Zamir, and Silvio Savarese. Action recognition by hierarchical mid-level action elements. *International Conference on Computer Vision*, 2015.

[135] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[136] Michelle Lee*, Yuke Zhu*, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. *International Conference on Robotics and Automation*, 2019.

[137] Scott Lenser and Manuela Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *International Conference on Intelligent Robots and Systems*, 2003.

[138] Adam Lerer, Sam Gross, and Rob Fergus. Learning physical intuition of block towers by example. In *International Conference on Machine Learning*, 2016.

[139] Timothée Lesort, Natalia Díaz Rodríguez, Jean-Franois Goudou, and David Filliat. State representation learning for control: An overview. *arXiv preprint arXiv:1802.04181*, 2018.

[140] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 2016.

[141] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[142] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Symposium on Experimental Robotics*, 2016.

[143] Wenbin Li, Seyedmajid Azimi, Ales Leonardis, and Mario Fritz. To fall or not to fall: A visual approach to physical stability prediction. *arXiv preprint arXiv:1604.00066*, 2016.

[144] Yunzhu Li, Jiaming Song, and Stefano Ermon. Inferring the latent structure of human decision-making from raw visual inputs. *Advances in Neural Information Processing Systems*, 2017.

[145] Yitao Liang, Marlos Machado, Erik Talvitie, and Michael Bowling. State of the art control of atari games using shallow reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, 2016.

[146] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

[147] Chris Linegar, Winston Churchill, and Paul Newman. Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera. In *International Conference on Robotics and Automation*, 2016.

[148] Changsong Liu, Shaohua Yang, Sari Saba-Sadiya, Nishant Shukla, Yunzhong He, Song-Chun Zhu, and Joyce Chai. Jointly learning grounded task structures from language instruction and visual demonstration. In *Empirical Methods in Natural Language Processing*, 2016.

[149] Guan-Horng Liu, Avinash Siravuru, Sai Prabhakar, Manuela Veloso, and George Kantor. Learning end-to-end multimodal sensor policies for autonomous navigation. *arXiv preprint arXiv:1705.10422*, 2017.

[150] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *International Conference on Robotics and Automation*, 2018.

[151] Minh-Thang Luong, Hieu Pham, and Christopher Manning. Effective approaches to attention-based neural machine translation. In *Empirical Methods in Natural Language Processing*, 2015.

[152] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[153] Mohsen Malmir, Karan Sikka, Deborah Forster, Javier Movellan, and Garison Cottrell. Deep Q-learning for active recognition of germs: Baseline performance on a standardized dataset for active learning. In *British Machine Vision Conference*, 2015.

[154] Ajay Mandlekar, Jonathan Booher, Max Spero, Albert Tung, Anchit Gupta, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity. In *International Conference on Intelligent Robots and Systems*, 2019.

[155] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. *Conference on Robot Learning*, 2018.

[156] Javier Marin, David Vázquez, David Gerónimo, and Antonio López. Learning appearance in virtual scenarios for pedestrian detection. In *Conference on Computer Vision and Pattern Recognition*, 2010.

[157] Colin McManus, Ben Upcroft, and Paul Newmann. Scene signatures: Localised and pointless features for localisation. In *Robotics: Science and Systems*, 2014.

[158] Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg

Wayne, and Nicolas Heess. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201*, 2017.

[159] Jeff Michels, Ashutosh Saxena, and Andrew Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine Learning*, 2005.

[160] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[161] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[162] Roozbeh Mottaghi, Hessam Bagherinezhad, Mohammad Rastegari, and Ali Farhadi. Newtonian image understanding: Unfolding the dynamics of objects in static images. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[163] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav Gupta, and Ali Farhadi. "what happens if..." learning to predict the effect of forces in images. In *European Conference on Computer Vision*, 2016.

[164] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. *arXiv preprint arXiv:1709.10089*, 2017.

[165] Dana Nau, Yue Cao, Amnon Lotem, and Hector Munoz-Avila. Shop: Simple hierarchical ordered planner. In *International Joint Conferences on Artificial Intelligence*, 1999.

[166] Andrew Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287, 1999.

[167] Andrew Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

[168] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Ng. Multimodal deep learning. In *International Conference on Machine Learning*, pages 689–696, 2011.

[169] Monica Nicolescu and Maja Matarić. A hierarchical architecture for behavior-based robots. In *International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 227–233, 2002.

[170] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G Barto. Learning and generalization of complex tasks from unstructured demonstrations. In *International Conference on Intelligent Robots and Systems*, 2012.

[171] Alva Noë and J. Kevin O'Regan. On the brain-basis of visual consciousness: A sensorimotor account. *Vision and mind: Selected readings in the philosophy of perception*, 2002.

[172] Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in ATARI games. In *Advances in Neural Information Processing Systems*, pages 2863–2871, 2015.

[173] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[174] Giuseppe Oriolo, Marilena Vendittelli, and Giovanni Ulivi. On-line map building and navigation for autonomous mobile robots. In *International Conference on Robotics and Automation*, 1995.

[175] Andrew Owens and Alexei Efros. Audio-visual scene analysis with self-supervised multisensory features. *European Conference on Computer Vision*, 2018.

[176] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2009.

[177] Jeremie Papon and Markus Schoeler. Semantic pose using deep networks trained on synthetic RGB-D. In *International Conference on Computer Vision*, 2015.

[178] Emilio Parisotto, Lei Jimmy Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *International Conference on Learning Representations*, 2016.

[179] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, 1998.

[180] Peter Pastor, Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, and Stefan Schaal. Skill learning and task outcome prediction for manipulation. In *IEEE International Conference on Robotics and Automation*, 2011.

[181] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-Real Transfer of Robotic Control with Dynamics Randomization. *arXiv preprint arXiv:1710.06537*, October 2017.

[182] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *International Conference on Robotics and Automation*, pages 1–8, 2018.

[183] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 2008.

[184] Stephen Phillips, Andrew Jaegle, and Kostas Daniilidis. Fast, robust, continuous monocular egomotion computation. In *International Conference on Robotics and Automation*, 2016.

[185] Jean Piaget. *The development of thought: Equilibration of cognitive structures*. Viking, 1977.

[186] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *Robotics: Science and Systems*, 2018.

[187] Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and Abhinav Gupta. The curious robot: Learning visual representations via physical interactions. In *European Conference on Computer Vision*, 2016.

[188] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *International Conference on Robotics and Automation*, 2016.

[189] Brahayam Ponton, Alexander Herzog, Stefan Schaal, and Ludovic Righetti. A convex model of humanoid momentum dynamics for multi-contact motion generation. In *International Conference on Humanoid Robots*, pages 842–849, 2016.

[190] Ivaylo Popov, Nicolas Heess, Timothy Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*, 2017.

[191] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *International Journal of Robotics Research*, 33(7):1044–1044, June 2014.

[192] Rouhollah Rahmatizadeh, Pooya Abolghasemi, Ladislau Bölöni, and Sergey Levine. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. *arXiv preprint arXiv:1707.02920*, 2017.

[193] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[194] Scott Reed and Nando de Freitas. Neural programmer-interpreters. In *International Conference on Learning Representations*, 2016.

[195] Anthony Remazeilles, François Chaumette, and Patrick Gros. Robot motion control from a visual memory. In *International Conference on Robotics and Automation*, 2004.

[196] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015.

[197] Stephan Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In *European Conference on Computer Vision*, 2016.

[198] Ludovic Righetti, Mrinal Kalakrishnan, Peter Pastor, Jonathan Binney, Jonathan Kelly, Randolph Voorhies, Gaurav Sukhatme, and Stefan Schaal. An autonomous manipulation system based on force control and optimization. *Autonomous Robots*, 36(1):11–30, 2014.

[199] Joseph Romano, Kaijen Hsiao, Günter Niemeyer, Sachin Chitta, and Katherine Kuchenbecker. Human-inspired robotic grasp control with tactile sensing. *IEEE Transactions on Robotics*, 27(6):1067–1079, 2011.

[200] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[201] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, pages 627–635, 2011.

[202] Eric Royer, Jonathan Bom, Michel Dhome, Benoit Thuilot, Maxime Lhuillier, and Franois Marmoiton. Outdoor autonomous navigation using monocular vision. In *International Conference on Intelligent Robots and Systems*, 2005.

[203] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

[204] Andrei Rusu, Neil Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[205] Andrei Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

[206] Andrei A. Rusu, Sergio Gomez Colmenarejo, Çaglar Gülçehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *International Conference on Learning Representations*, 2016.

[207] Inkyu Sa, Stefan Hrabar, and Peter Corke. Inspection of pole-like structures using a vision-controlled vtol uav and shared autonomy. In *International Conference on Intelligent Robots and Systems*, 2014.

[208] Earl Sacerdoti. A structure for plans and behavior. Technical report, SRI International's Artificial Intelligence Center, 1975.

[209] Parvaneh Saeedi, Peter Lawrence, and David Lowe. Vision-based 3-d trajectory tracking for unknown environments. *IEEE Transactions on Robotics*, 2006.

[210] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, 2015.

[211] Connor Schenck and Dieter Fox. Reasoning about liquids via closed-loop simulation. *arXiv preprint arXiv:1703.01656*, 2017.

[212] Jürgen Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *International Joint Conference on Neural Networks*, 1990.

[213] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[214] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[215] Siddarth Sen*, Animesh Garg*, David Gealy, Stephen McKinley, Yiming Jen, and Ken Goldberg. Autonomous Multiple-Throw Multilateral Surgical Suturing with a Mechanical Needle Guide and Optimization based Needle Planning. In *International Conference on Robotics and Automation*, 2016.

[216] Ozan Sener, Amir Zamir, Silvio Savarese, and Ashutosh Saxena. Unsupervised semantic parsing of video collections. In *International Conference on Computer Vision*, 2015.

[217] Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-contrastive networks: Self-supervised learning from multi-view observation. *arXiv preprint arXiv:1704.06888*, 2017.

[218] Alireza Shafaei, James Little, and Mark Schmidt. Play and learn: using video games to train computer vision models. *British Machine Vision Conference*, 2016.

[219] David Silver, Aja Huang, Chris Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[220] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, 2014.

[221] Robert Sim and James Little. Autonomous vision-based exploration and mapping using hybrid maps and rao-blackwellised particle filters. In *International Conference on Intelligent Robots and Systems*, 2006.

[222] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2014.

[223] Jivko Sinapov, Connor Schenck, and Alexander Stoytchev. Learning relational object categories using behavioral exploration and multimodal perception. In *IEEE International Conference on Robotics and Automation*, pages 5691–5698, 2014.

[224] Hee Chan Song, Young Loul Kim, and Jae-Bok Song. Automated guidance of peg-in-hole assembly tasks for complex-shaped parts. In *International Conference on Intelligent Robots and Systems*, pages 4517–4522, 2014.

[225] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2222–2230, 2012.

[226] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *International Conference on Robotics and Automation*, 2014.

[227] Siddharth Srivastava, Lorenzo Riano, Stuart Russell, and Pieter Abbeel. Using classical planners for tasks with continuous operators in robotics. In *International Conference on Automated Planning and Scheduling*, 2013.

[228] Zhe Su, Karol Hausman, Yevgen Chebotar, Artem Molchanov, Gerald E. Loeb, Gaurav Sukhatme, and Stefan Schaal. Force estimation and slip detection/classification for grip control using a biomimetic tactile sensor. In *International Conference on Humanoid Robots*, pages 297–303, 2015.

[229] Jaeyong Sung, Kenneth Salisbury, and Ashutosh Saxena. Learning to represent haptic feedback for partially-observable tasks. In *IEEE International Conference on Robotics and Automation*, pages 2802–2809, 2017.

[230] Jaeyong Sung, Bart Selman, and Ashutosh Saxena. Learning sequences of controllers for complex manipulation tasks. In *International Conference on Machine Learning*, 2013.

[231] Richard Sutton and Andrew Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[232] Richard Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[233] Aviv Tamar, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2016.

[234] Makarand Tapaswi, Martin Bauml, and Rainer Stiefelhagen. Book2movie: Aligning video scenes with book chapters. In *Conference on Computer Vision and Pattern Recognition*, 2015.

[235] Matthew Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(July):1633–1685, 2009.

[236] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *International Conference on Intelligent Robots and Systems*, 2017.

[237] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[238] Masahiro Tomono. 3-D Object Map Building Using Dense Object Models with SIFT-Based Recognition Features. In *International Conference on Intelligent Robots and Systems*, 2006.

[239] Steve Tonneau, Andrea Del Prete, Julien Pettré, Chonhyon Park, Dinesh Manocha, and Nicolas Mansard. An efficient acyclic contact planner for multiped robots. *IEEE Transactions on Robotics*, 34(3):586–601, 2018.

[240] Son Tran and Larry Davis. Event modeling and recognition using markov logic networks. In *European Conference on Computer Vision*, 2008.

[241] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 2008.

[242] Herke van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *International Conference on Intelligent Robots and Systems*, pages 3928–3934, 2016.

[243] Herke Van Hoof, Tucker Hermans, Gerhard Neumann, and Jan Peters. Learning robot in-hand manipulation with tactile features. In *International Conference on Humanoid Robots*, pages 121–127, 2015.

[244] Matej Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[245] Filipe Veiga, Herke Van Hoof, Jan Peters, and Tucker Hermans. Stabilizing novel objects by learning to predict tactile slip. In *International Conference on Intelligent Robots and Systems*, pages 5065–5072, 2015.

[246] Ulrich Viereck, Andreas ten Pas, Kate Saenko, and Robert Platt. Learning a visuomotor controller for real world robotic grasping using easily simulated depth images. *arXiv preprint arXiv:1706.04652*, 2017.

[247] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 2002.

[248] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[249] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion. *Conference on Computer Vision and Pattern Recognition*, 2019.

[250] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018.

[251] Xiaolong Wang, Ali Farhadi, and Abhinav Gupta. Actions ˜ transformations. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[252] Ziyu Wang, Josh Merel, Scott Reed, Greg Wayne, Nando de Freitas, and Nicolas Heess. Robust imitation of diverse behaviors. *Advances in Neural Information Processing Systems*, 2017.

[253] Daniel Whitney. Quasi-Static Assembly of Compliantly Supported Rigid Parts. *Journal of Dynamic Systems, Measurement, and Control*, 104(1):65–77, 1982.

[254] Daniel Whitney. Historical perspective and state of the art in robot force control. *International Journal of Robotics Research*, 6(1):3–14, 1987.

[255] David Whitney, Eric Rosen, Elizabeth Phillips, George Konidaris, and Stefanie Tellex. Comparing Robot Grasping Teleoperation across Desktop and Virtual Reality with ROS Reality. In *International Symposium on Robotics Research*, 2017.

[256] David Wooden. A guide to vision-based map building. *IEEE Robotics and Automation Magazine*, 2006.

[257] Jiajun Wu, Ilker Yildirim, Joseph Lim, William Freeman, and Joshua B. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. In *Advances in Neural Information Processing Systems*, 2015.

[258] Yan Wu and Yiannis Demiris. Towards one shot learning by imitation for humanoid robots. In *International Conference on Robotics and Automation*, 2010.

[259] Bernhard Wymann, Christos Dimitrakakis, Andrew Sumner, Eric Espié, and Christophe Guionneau. TORCS: The open racing car simulator. `http://www.torcs.org`, 2015.

[260] Danfei Xu*, Suraj Nair*, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *International Conference on Robotics and Automation*, 2018.

[261] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene graph generation by iterative message passing. *Conference on Computer Vision and Pattern Recognition*, 2017.

[262] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. *arXiv preprint arXiv:1610.00673*, 2016.

[263] Xitong Yang, Palghat Ramesh, Radha Chitta, Sriganesh Madhvanath, Edgar A Bernal, and Jiebo Luo. Deep multimodal representation learning from temporal data. In *Conference on Computer Vision and Pattern Recognition*, pages 5066–5074, 2017.

[264] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot imitation from observing humans via domain-adaptive meta-learning. *Robotics: Science and Systems*, 2018.

[265] Matthew Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, 2014.

[266] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *arXiv preprint arXiv:1710.04615*, 2017.

[267] Yuke Zhu, Alireza Fathi, and Li Fei-Fei. Reasoning about object affordances in a knowledge base representation. *European Conference on Computer Vision*, 2014.

[268] Yuke Zhu*, Daniel Gordon*, Eric Kolve, Dieter Fox, Li Fei-Fei, Abhinav Gupta, Roozbeh Mottaghi, and Ali Farhadi. Visual semantic planning using deep successor representations. In *International Conference on Computer Vision*, 2017.

[269] Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. Visual7W: Grounded Question Answering in Images. In *Conference on Computer Vision and Pattern Recognition*, 2016.

[270] Yuke Zhu, Joseph Lim, and Li Fei-Fei. Knowledge acquisition for visual question answering via iterative querying. *Conference on Computer Vision and Pattern Recognition*, 2017.

[271] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *International Conference on Robotics and Automation*, 2017.

[272] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and imitation learning for diverse visuomotor skills. *Robotics: Science and Systems*, 2018.

[273] Yuke Zhu, Ce Zhang, Christopher Ré, and Li Fei-Fei. Building a large-scale multimodal knowledge base system for answering visual queries. *arXiv preprint arXiv:1507.05670*, 2015.

[274] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *International Conference on Computer Vision*, 2015.