Robots that can see : Learning visually guided behavior

Alexander Pashevich

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Alexander PASHEVICH

Thèse dirigée par **Cordelia SCHMID**,directeur de recherche, Université Grenoble Alpes

préparée au sein du **Laboratoire Laboratoire Jean Kuntzmann** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Des robots qui voient : Apprentissage de comportements guidés par la vision

Robots that can see: Learning visually guided behavior

Thèse soutenue publiquement le **29 septembre 2021**, devant le jury composé de :

Madame CORDELIA SCHMID
DIRECTEUR DE RECHERCHE, INRIA CENTRE GRENOBLE-RHONE-ALPES, Directrice de thèse
Monsieur CHRISTIAN WOLF
MAITRE DE CONFERENCE HDR, INSTITUT NATIONAL SC. APPLIQUEES DE LYON, Rapporteur
Monsieur NICOLAS MANSARD
DIRECTEUR DE RECHERCHE, CNRS DELEGATION OCCITANIE OUEST, Rapporteur
Monsieur IVAN LAPTEV
DIRECTEUR DE RECHERCHE, INRIA CENTRE DE PARIS, Examinateur
Monsieur JEAN-PAUL LAUMOND
DIRECTEUR DE RECHERCHE EMERITE, CNRS DELEGATION OCCITANIE OUEST, Examinateur
Monsieur JAMES L CROWLEY
PROFESSEUR DES UNIVERSITES, GRENOBLE INP, Président
Monsieur PIERRE-YVES OUDEYER
DIRECTEUR DE RECHERCHE, INRIA CENTRE BORDEAUX SUD-OUEST, Examinateur

Abstract

Recently, vision and learning made significant progress that could improve robot control policies for complex environments. In this thesis, we introduce novel methods for learning robot control that improve the state-of-the-art on challenging tasks. We also propose a novel approach for the task of learning control in dynamic environments guided by natural language.

Data availability is one of the major challenges for learning-based methods in robotics. While collecting a dataset from real robots is expensive and limits scalability, simulators provide an attractive alternative. Policies learned in simulation, however, usually do not transfer well to real scenes due to the domain gap between real and synthetic data. We propose a method that enables task-independent policy learning for real robots using only synthetic data. We demonstrate that our approach achieves excellent results on a range of real-world manipulation tasks.

Learning-based approaches can solve complex tasks directly from camera images but require non-trivial domain-specific knowledge for their supervision. This thesis introduces two novel methods for learning visually guided control policies given a limited amount of supervision. First, we propose a reinforcement learning approach that learns to combine skills using neither intermediate rewards nor complete task demonstrations. Second, we propose a new method to solve a task specified with a solution example employing a novel disassembly procedure. While using no real images for training, we demonstrate the versatility of our methods in challenging real-world settings including temporary occlusions and dynamic scene changes.

Interaction and navigation defined by natural language instructions in dynamic environments pose significant challenges for learning-based methods. To handle long sequences of subtasks, we propose a novel method based on a multimodal transformer that encodes the full history of observations and actions. We also propose to leverage synthetic instructions as intermediate representations to improve understanding of complex human instructions.

For all the contributions, we validate our approaches against strong baselines and show that they outperform previous state-of-the-art methods.

Keywords: robotics, reinforcement learning, imitation learning, sim2real transfer, visual-and-language navigation, neural networks, computer vision, natural language processing, machine learning.

Résumé

Récemment, la vision par ordinateur et l'apprentissage automatique ont fait des progrès significatifs qui pourraient améliorer le contrôle des robots dans les environnements complexes. Dans ce manuscrit, nous introduisons de nouvelles méthodes d'apprentissage de comportements des robots. Nous proposons également une nouvelle approche pour la tâche d'apprentissage du contrôle guidé par le langage naturel.

La disponibilité des données reste l'un des défis principaux pour les méthodes d'apprentissage en robotique. Néanmoins, bien que la collecte d'un ensemble de données à partir de robots réels soit coûteuse et rarement extensible, les simulateurs offrent aujourd'hui une alternative attrayante. Le comportement appris en simulation, cependant, ne se transfère généralement pas adéquatement à les scènes réelles à cause de la difference principale entre les données réelles et synthétiques. Pour faire face à cette limitation, nous proposons dans cette thèse une méthode qui permet un apprentissage de comportements pour les robots réels en utilisant uniquement des données synthétiques. Nous démontrons que notre approche aboutit à d'excellents résultats sur une gamme de tâches de manipulation dans un milieu réel.

Les approches d'apprentissage peuvent résoudre des tâches complexes directement à base des images, mais nécessitent des connaissances spécifiques au domaine pour leur supervision. Nous proposons deux méthodes d'apprentissage des comportements guidés par la vision compte tenu d'une supervision limitée. Premièrement, nous proposons une approche d'apprentissage par renforcement qui apprend à combiner des compétences primitives. Deuxièmement, nous proposons une nouvelle méthode pour résoudre des tâches définies avec un exemple de solution qui utilise une procédure innovante de désassemblage. Nous démontrons la polyvalence de nos méthodes dans des contextes réels complexes, y compris des occlusions et des changements dynamiques.

L'interaction et la navigation définies par le langage naturel dans des environnements dynamiques posent des défis importants pour les méthodes d'apprentissage. Pour gérer une longue séquence de sous-tâches, nous proposons une nouvelle méthode qui garde l'historique complet des observations et des actions. Nous proposons également d'utiliser des instructions synthétiques pour améliorer la compréhension des instructions humaines complexes.

Pour toutes nos contributions, nous avons comparé nos approches avec les techniques existantes et nous montrons que nos résultats sont significativement meilleurs que ceux de l'état de l'art.

Acknowledgements

The accomplishment of this thesis would not have been possible without the help of numerous people. First of all, I would like to thank my advisor, Cordelia Schmid, for her knowledge, energy, support, and patience. I am forever grateful for your guidance and time throughout my PhD. I am very thankful to all the jury members for accepting to evaluate my work and in particular, to Christian Wolf and Nicolas Mansard for taking the time to review this manuscript.

Thoth and Willow teams at Inria provided me with an amazing environment for doing research and exceptional people to learn from. I am sincerely grateful to Ivan Laptev who co-advised me on several projects for his ideas and commitment. I would like to thank my collaborators at Willow, Robin Strudel and Igor Kalevatykh, for a great teamwork experience. I was fortunate to be a part of Ganesha team at Google for six months where I was warmly received by all team members. I am especially thankful to Chen Sun, my internship host, for his kindness and amazing supervision. I would like to also thank Danijar Hafner, James Davidson, and Rahul Sukthankar who I had a pleasure to work with on my first project. I am very grateful to Radu Horaud, my first supervisor at Inria, for showing me the beauty of research and to Lidia Beliovskaya, my first teacher of informatics and robotics, for instilling a passion to the field in me. I am very thankful to Nathalie for being extremely responsive during my whole stay at Inria. I also thank both Dashas, Lina, Kostya, and Charles for proofreading my manuscript.

The list of colleagues and friends that have made my PhD an amazing journey is long, and I hope those I missed will forgive me. To members of Thoth team: Nikita for mentoring me during my first years, Konstantin for answering an incredible amount of questions, Alberto for always useful scientific discussions, Adria, Thomas, and Vicky for being great office mates, Ricardo for helping me with the cluster, Valentin for sharing useful insights. To members of Willow team who made me feel welcome in Paris: Robin, Igor, Gul, Pierre-Louis, Justin, and Gregoire. To Michel and Maria, interns at Thoth who I worked with. To my amazing friends all over France: Kostya, Nikita, Jack, Dasha, Ahmet, Lina, Valentin, Fabien, Vitalii, Camilo, Tanya, Minttu, Anurag, two Ivans, Anya, and Kyriakos.

I would like to express infinite gratitude to my father and mother for their love and the education they gave me. I will be forever grateful to France for offering me scholarships to make my studies here possible. Finally, last but not least, I would like to thank my dear Dasha who I met in this PhD journey for her love and support.

Contents

1

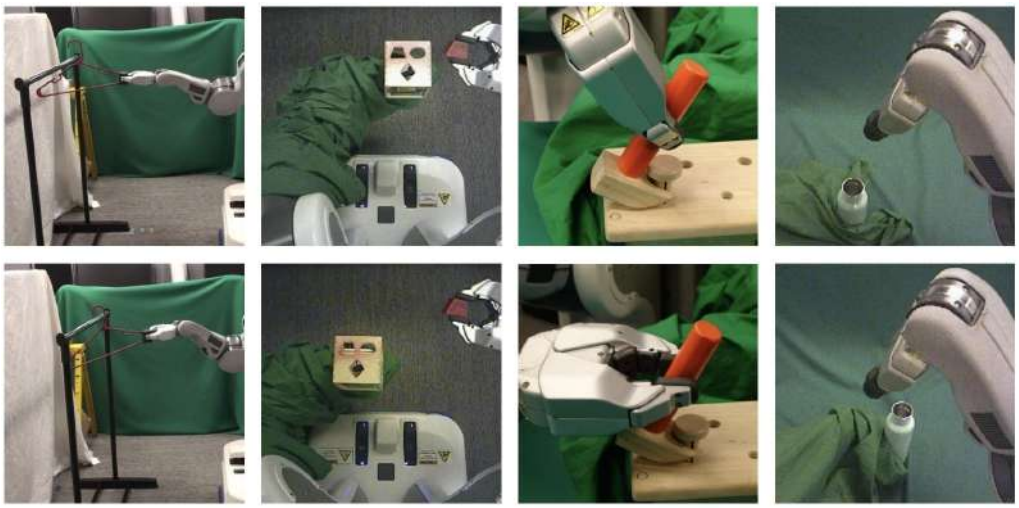
Introduction

"... spend the summer linking a camera to a computer and getting the computer to describe what it saw."

Marvin Minsky on the goal of a 1966 undergraduate summer research project [Boden, 2008]

Recent advances in machine learning offer an opportunity to build intelligent systems with a wide range of capacities. Machine learning is a class of methods to find patterns and regularities in collected datasets which are then used to make predictions and decisions. Deep learning is a branch of machine learning where artificial neural networks with multiple layers are employed as function approximators. Parameters of the neural networks are learned using stochastic gradient descent methods which minimize a loss function chosen for a specific task.

The history of deep learning goes back to the 1950s [Rosenblatt, 1957, LeCun et al., 1989, Rumelhart et al., 1986]. However, it received a new level of attention in 2012 when neural networks first time outperformed earlier models by a large margin on an image classification task [Krizhevsky et al., 2012]. Since 2012, neural networks set new state-of-the-art results in many areas including image and video understanding [He et al., 2017, Simonyan and Zisserman, 2014a], image and text generation [Goodfellow et al., 2014, Brown et al., 2020], speech recognition [Dahl et al., 2012], game playing [Silver et al., 2016], machine translation [Wu et al., 2016], and many others. The research community demonstrated that given a large corpus of data, powerful compute resources, and very expressive models, a deep learning approach can outperform models hand-engineered by domain experts.



(a) hanger (b) cube (c) hammer (d) bottle

Figure 1.1 – Example of tasks where deep reinforcement learning was successfully applied [Levine et al., 2015].

The robotics community adopted deep learning techniques and used them to improve robotic perception and control. One popular application is employing deep neural networks for object recognition, pose estimation, or scene understanding and providing network outputs to a downstream control system [Labbe et al., 2020, Andrychowicz et al., 2017, Finn et al., 2016]. Another application is explicitly using deep neural networks for control policies learning and decision making [Zhu et al., 2017, Levine et al., 2015] illustrated in Figure 1.1. Reinforcement learning (RL) is a popular approach to learn control explicitely where the goal is to train a policy maximizing rewards [Sutton and Barto, 1998]. The RL setting assumes that an agent interacts with the world using a trial-and-error approach and receives feedback defined with a scalar reward function. Deep reinforcement learning (Deep RL) uses the same algorithms as RL but employs deep neural networks as function approximators [Mnih et al., 2013]. Another popular approach to learn control is imitation learning (IL) where an agent is provided with demonstrations of successful task execution from an expert [Michie et al., 1990, Atkeson and Schaal, 1997].

One of the major challenges in robotics now is designing systems which work in a wide range of conditions while using as little expert knowledge about a specific task as possible. A real-world example of the ideal scenario would be a household robot that can help humans to perform routine tasks as soon as it is placed in an unknown apartment. Our work is a step towards

Figure 1.2 – Collecting large-scale datasets for robotics may require thousands of hours of interaction and can be very expensive in practice [Levine et al., 2016].

solving this challenge where we propose novel approaches to learn control for real-world robots with the help of simulators. Our approaches are designed to use less task-specific expertise than previously proposed methods and to be able to operate under dynamical changes in a robot scene. We also propose an approach to control autonomous agents using natural language instructions.

1.1 Goals

The goal of our work is to propose novel methods to learn control policies for real-world robotic systems which are able to perform a wide range of tasks. We aim at designing robust systems that can work under dynamic changes in their scene. We also make a step towards enabling autonomous agents to understand instructions given in natural language.

The essential part of any deep learning approach is data to learn from. Designing a robotic system which learns from a large scale real-world dataset requires a mechanism for data labeling and automatic resets of the scene [Zeng et al., 2019]. Even if such mechanism is implemented, collecting large amounts of domain-specific data limits the scalability as learning policies typically requires hundreds or thousands of hours of interaction [Levine et al., 2016, Pinto and Gupta, 2016] (see Figure 1.2). Moreover, the dataset might need to be recollected once scene conditions such as lighting, background clutter, or a camera configuration change. The mentioned factors

Figure 1.3 – An example of a vision-and-language navigation (VLN) task [Shridhar et al., 2020] where an agent is required to achieve a goal specified by natural language.

make learning control policies from real-world data challenging. Instead, we aim to learn control policies using data from a physics simulator where the real-world robotic scene is mimicked. Such data collection can be easily parallelized and collected cheaply on scale [Liang et al., 2018]. We propose to make learned control policies have similar behavior in simulated and real scenes by finding a simulation-to-reality (sim2real) transformation and applying it to synthetic images used for training. We also use this transformation to account for changing factors of the robotic scene such as the camera configuration and scene object shapes.

Learning a control policy through trial-and-error requires careful definition of the reward function which is called reward engineering. Specifying the reward function becomes relatively straightforward once it is done using only the completion signal. For example, a navigation agent can only be provided with a signal whether it reached the target or not. In this case, the supervision of a robotic system can be provided without domain-specific knowledge and is therefore cheaper. However, in contrast to humans that

can achieve goals over an entire lifetime with little to no reward, RL algorithms often struggle to find solutions under such weak supervision [Nair et al., 2018, Pathak et al., 2017]. To learn complex behavior from completion signals, we propose to use a hierarchical architecture where low-level policies called skills are pretrained using short demonstrations and a high-level policy learns to combine the skills through trial-and-error using the completion signal. An alternative for providing supervision with little domain expertise is showing an agent an example of a completed task. In the case of a robotic manipulator assembling furniture, the agent can be shown an example of an assembled item. For this setting, we propose to reverse the problem by disassembling the given example first and treating the disassembly action sequences as a reversed assembly demonstration.

The most natural way of specifying a task for a robot is nevertheless the human language. In this case, a robotic agent needs to reason about natural language instructions and ground them on the perception of other sensors. Such problems are known as multimodal and represent a significant challenge for learnable agents. In this dissertation, we take a step towards bridging robotics and natural language processing and address a visual-and-language navigation (VLN) task. The VLN task assumes that an agent is placed in a simulated environment and is given a textual command defining its goal. An example of a VLN problem is shown in Figure 1.3 where an agent is required to navigate and interact in a dynamic environment. The VLN setting requires an agent to walk in a partially observable environment and to have long-term memory about its previous decisions and observations. Recently, an attention-based architecture called transformer [Vaswani et al., 2017] was shown to improve state-of-the-art results on tasks that require handling long sequences such as language processing [Devlin et al., 2019, Sun et al., 2019]. We propose to employ the transformer architecture for sequential decision making on the multimodal VLN task and show its advantages over prior work.

1.2 Context

Deep reinforcement learning (deep RL) is a popular approach for autonomous agent training due to its ability to acquire complex behavior using high-dimensional sensory input [Mnih et al., 2013] and supervision in a form of a scalar reward function [Silver et al., 2017]. In robotics where manual controller design can be difficult and highly specific for a task and a robot platform, those properties make deep RL especially appealing for the community. Although deep RL was successfully used to

Figure 1.4 – A human teleoperator is using virtual reality to record expert demonstrations for a robot [Zhang et al., 2018].

solve challenging problems in simulation [Berner et al., 2019, Jaderberg et al., 2017, Heess et al., 2017, Peng et al., 2016], its application to real-world problems is not straightforward due to the amount of required data. The real-world applications of deep RL were mostly limited to locomotion tasks with stable robots [Ha et al., 2018], simple manipulation tasks [Vecerik et al., 2017, Mahmood et al., 2018], or tasks defined using low-dimensional reparametrizations [Calandra et al., 2015].

Another popular approach for training autonomous agents is imitation learning (IL) where an agent learns a policy given demonstrations of the desired behavior [Morales and Sammut, 2004, Ratliff et al., 2008, Duan et al., 2017]. Similar to Deep RL, IL algorithms were shown to work successfully with deep neural networks used as function approximators [Pan et al., 2018, Ho and Ermon, 2016]. One of the most popular methods in IL is behavior cloning (BC) [Pomerleau, 1989] which learns to explicitely map observed inputs into desired control outputs given a collected dataset [Bojarski et al., 2017, Chen and Huang, 2017]. While providing demonstrations is feasible for a real robot in some cases, for example, using teleoperation [Zhang et al., 2018] as shown in Figure 1.4, the demonstrations are usually more expensive to obtain than trials used in RL. Moreover, the IL methods and BC, in particular, are known to suffer from the errors accumulating over time known as the compounding errors problem [Xu et al., 2020]. One of the ways to address the compounding errors is to use a feedback from an online expert [Ross and Bagnell, 2014] which however requires additional domain expertise. Another principled approach is injecting noise into expert demonstrations [Laskey et al., 2017]. In our work, we explore both RL

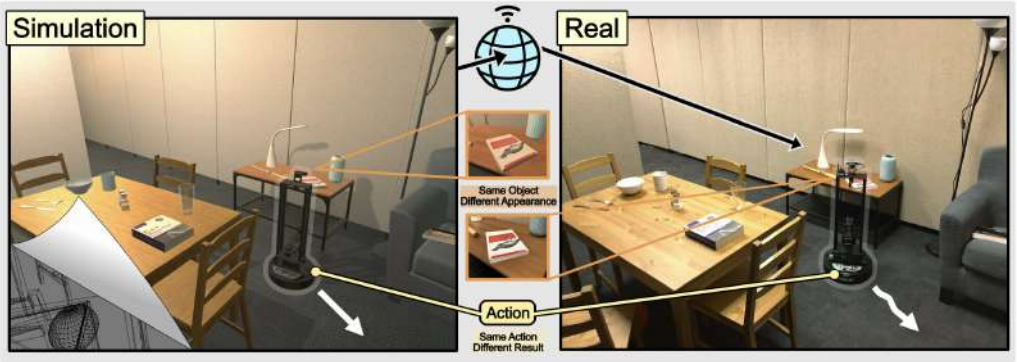


Figure 1.5 – RoboTHOR is an example of a physics simulator that features simulation to reality pairings [Deitke et al., 2020].

and IL approaches and propose novel ways of combining their strengths.

Data availability is a fundamental challenge of applying deep learning techniques to train real-world autonomous agents. Modern deep learning methods in computer vision and natural language processing use offline datasets that contain several dozens or even hundreds of millions of labeled examples [Sun et al., 2017, Wu et al., 2016]. Obtaining such datasets for robotics is challenging due to the cost of robots, their exploitation time, difficulties to label data and automatically reset a robotic scene [Zhong et al., 2017]. Therefore, one of the major research directions in robot learning is developing methods to overcome the data requirement. Pretraining a neural network on a large general-purpose dataset was shown to be efficient for reducing data requirements of a downstream task learning in the context of supervised learning [Devlin et al., 2019, Shelhamer et al., 2017]. Recently, similar techniques were used in the context of RL and IL methods [Hao et al., 2020, Gupta et al., 2018]. Another direction that holds the promise of lower data requirements is model-based RL which aims to aid learning by introducing knowledge about the world [Hafner et al., 2019, Levine et al., 2015]. With the same itention of reducing the data requirements, meta reinforcement learning (meta RL) methods learn to solve a distribution of tasks to transfer this knowledge to a downstream task [Duan et al., 2016, Finn et al., 2017].

Learning to control a robot using synthetic data from a physics simulator and transferring such policies to the real world is an alternative solution that we consider in our work. Realistic simulators provide a cheap and scalable way of collecting enough trials and demonstrations to train deep neural networks [Todorov et al., 2012, Coumans, 2009]. Another advantage of simulators is the fully observable state of the world that can be

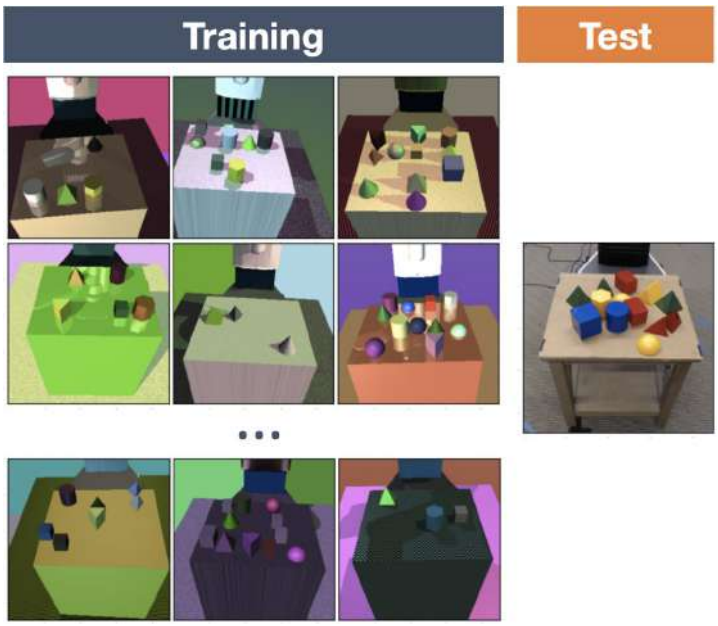


Figure 1.6 – Data randomization is an idea of randomizing simulated data used for training to make learned policies to work in real-world scenes without finetuning [Tobin et al., 2017].

accessed at any time and used for label generation without the need for human annotators [Shridhar et al., 2020, Anderson et al., 2018, Liang et al., 2018]. The main challenge of using synthetic data for learning real-world policies is the difference between simulation and reality which is known as simulation-to-reality (sim2real) gap [Jakobi et al., 1995]. This difference is observed in both outputs of simulated sensors and the physical behavior of simulated objects. As a result, the policies trained on synthetic data often decrease their performance when used in the real world [Tan et al., 2018]. Despite a lot of recent effort of improving simulators to better match the reality [Deitke et al., 2020, Kolve et al., 2017] (see Figure 1.5), having a fast and realistic simulation remains an open challenge [Hennigh et al., 2020, Erez et al., 2015].

The two principled approaches to overcome the sim2real gap are data adaptation and data randomization. The data adaptation aims to improve the real-world performance of a learned policy by leveraging data from a simulator. Examples of data adaptation include pretraining on simulated data and finetuning on real data [Karttunen et al., 2020], learning only a specific part of the model using the real data [Mordatch et al., 2016], and making distributions of real and simulation domains to match on either sensor or feature level [Bousmalis et al., 2018, James et al., 2018, Lee et al.,

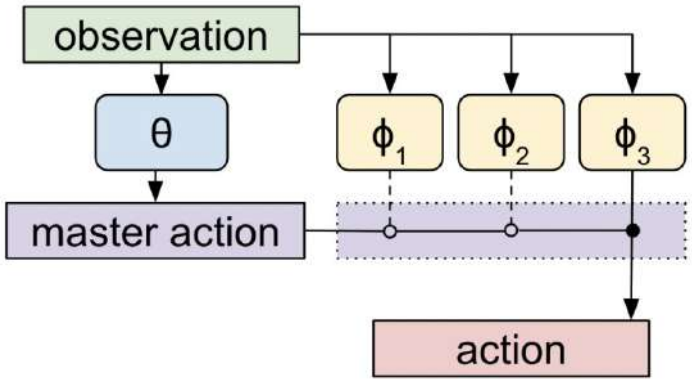


Figure 1.7 – Hierarchical RL assumes training several policies of different levels. This figure shows a two-level hierarchy of low-level policies (ϕ_1, ϕ_2, ϕ_3) which predict an action and a high-level policy θ which switches between them [Frans et al., 2018].

2018]. The data randomization uses simulated data to learn real-world policies by randomizing the simulation and ensuring that the data from a real scene falls in the randomized distribution [Tobin et al., 2017, Sadeghi and Levine, 2017] as shown in Figure 1.6. In this dissertation, we focus on the data randomization approach.

Learning a control policy with RL requires defining a reward function. While specifying rewards using several criteria of completion might seem fairly straightforward, such sparse rewards often introduce new local optima that can prevent agents from achieving optimal behavior [Jain et al., 2021, Hadfield-Menell et al., 2017, Trott et al., 2019]. Moreover, learning from sparse rewards may require a very large amount of data due to a much harder exploration problem [Bellemare et al., 2016]. Reward shaping is a popular technique to modify the reward signal to facilitate learning [Ng et al., 1999], but may require problem-specific domain expertise [Clark and Amodei, 2016]. Another approach of handling sparse rewards is to use exploration heuristics which help an agent to discover useful behavior [Burda et al., 2019, Pathak et al., 2017]. Alternative approaches also include relabeling the goal of a task for each trial [Andrychowicz et al., 2017], learning objectives that encourage diverse behavior [Haarnoja et al., 2017], and generating curriculums of starting states with increasing distances to the goal [Florensa et al., 2017b].

Another principled way of learning from sparse rewards is to employ hierarchies of policies. Such hierarchical architectures are known to aid exploration both semantically and temporally [Nachum et al., 2019]. Hierarchical

methods in RL are generally based on either feudal framework [Dayan and Hinton, 1993] or options framework [Sutton et al., 1999]. The feudal approaches learn a high-level policy that modulates a single low-level policy using a control signal [Haarnoja et al., 2018, Nachum et al., 2018, Vezhnevets et al., 2017, Kulkarni et al., 2016, Hausman et al., 2018]. The option methods learn a high-level policy that switches between several low-level policies called skills [Lee et al., 2019, Frans et al., 2018, Bacon et al., 2017, Florensa et al., 2017a] as shown in Figure 1.7. Hierarchical RL may also employ IL to pretrain some policies in the hierarchy [Das et al., 2018], or even use policies trained with IL in the hierarchy directly [Le et al., 2018]. In this dissertation, we consider the options framework and propose a novel way of combining RL with IL.

While specifying a task with rewards or demonstrations may seem attractive, the most natural way for humans to explain what needs to be done is language. Reasoning about natural language poses a significant challenge itself due to many factors including language differences, ambiguity, contextual information importance, everyday changes, and others [Allen, 1995]. However, recent advances in deep learning significantly improved the state-of-the-art in natural language processing (NLP) [Belinkov and Glass, 2019]. One of the major improvements in NLP was achieved by replacing recurrent networks which rely on hidden states [Hochreiter and Schmidhuber, 1997] with architectures computing attention to the whole input sequence called transformers [Vaswani et al., 2017] (see Figure 1.8). Pretraining on a large corpus of data from Wikipedia [Devlin et al., 2019] brought another significant improvement and became a common first step for many NLP tasks [Young et al., 2018].

Solving a task that requires language reasoning, scene understanding, and control learning represents an even more difficult problem. On top of the challenges specific for each of the three domains, the agent needs to deal with the multimodality of the input. Recently, multimodal problems combining vision and language were successfully addressed using transformers [Sun et al., 2019, Gabeur et al., 2020, Ding et al., 2020]. One of the problems that combine vision, language, and control is vision-and-language navigation (VLN) [Anderson et al., 2018, Chen et al., 2019a]. The VLN problem places an agent in a simulated environment and provides it with navigation instructions given in natural language. A particularly interesting scenario of VLN which brings it closer to robotics is ALFRED benchmark [Shridhar et al., 2020] where the agent is required to interact with a dynamic environment. In this dissertation, we propose a novel approach for ALFRED that makes a step towards bridging the gap between advances in robot learning and NLP.

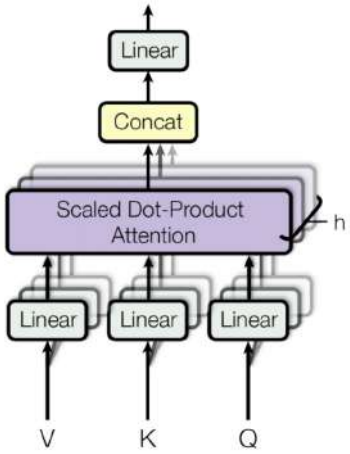


Figure 1.8 – The self-attention mechanism of transformer networks introduced in [Vaswani et al., 2017] improved state-of-the-art results in many domains.

1.3 Contributions

This thesis presents four main contributions. In Chapters 2-4, we propose novel methods for learning policies for real-world robots. In Chapter 5, we present an approach to solve a VLN problem defined with natural language instructions. The contributions of this thesis are summarized in the following paragraphs.

- To transfer control policies learned entirely from synthetic data to the real world, we propose an approach that learns an augmentation function making a policy behave similarly on simulated and real data. Given simulated and real domains and a set of random image transformations such as scaling and adding noise, our method finds a sequence of transformations and their parameters that are used to augment expert demonstrations in simulation. We show that the learned augmentation function is task-independent and can be used to learn control policies for several considered manipulation tasks. The first column of Figure 1.9 shows examples of depth images from expert demonstrations in simulation. The second column of Figure 1.9 shows the same images after applying the learned sim2real augmentation function to them. A control policy is trained to imitate the expert given the augmented synthetic images and is used directly on real-world images shown in the third column of Figure 1.9. We demonstrate a successful transfer of manipulation policies to real robot scenes for three tasks while using no real images for policy

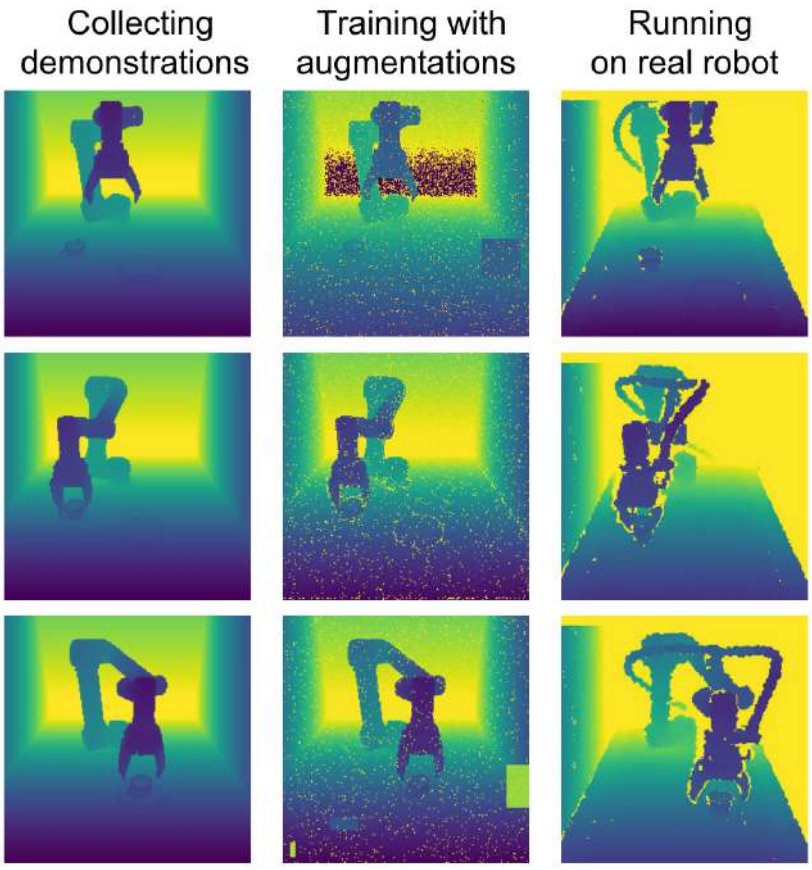


Figure 1.9 – Transferring a policy learned to place a cup in simulation to the real world. Synthetic depth images (first column) are augmented with random transformations during policy training (second column). The learned policy is applied to depth maps from the real robot scene (third column).

training. This work was published in IROS 2019 [Pashevich et al., 2019a] and is presented in Chapter 2. We also use the proposed approach in our later work presented in Chapters 3 and 4.

- To learn control policies for challenging manipulation tasks from sparse reward signals, we propose an approach that learns to combine skills. The skills such as grasping and going to are learned from few short synthetic demonstrations employing recent CNN architectures and data augmentation techniques. Given a vocabulary of pre-trained with IL skills, a high-level policy is trained to switch between the skill policies using RL as shown in Figure 1.10. In contrast to previously proposed methods, our approach does not require inter-

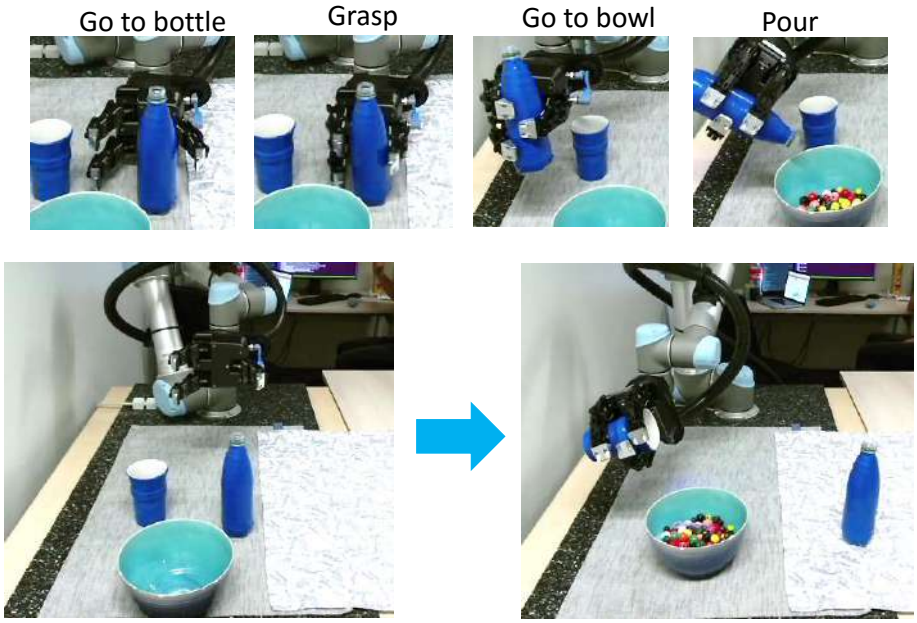


Figure 1.10 – A vocabulary of primitive robotics skills learned with Behavior Clonning is combined with Reinforcement Learning to perform a compositional task of grasping and pouring.

mediate rewards and complete task demonstrations during training. We also demonstrate the versatility of learned policies in challenging settings including temporary occlusions and dynamic scene changes. Notably, while all of our policies are learned on visual inputs in simulated images, we demonstrate a successful transfer to real-world scenes using the previously described sim2real approach. This work was published in ICRA 2020 [Strudel et al., 2020b] and is presented in Chapter 3.

- We propose a novel approach for learning control policies to build 3D objects specified by an example of their final configuration. The learned policies can assemble 3D shapes such as arches given building blocks of cubic and similar shapes shown in Figure 1.11 on the left. We propose to leverage a simulator to solve the problem in a low-dimensional state-space first and to transfer the learned policies to the high-dimensional image space of a real robot using the previously mentioned sim2real approach. We also propose a disassembling procedure which automatically generates a dataset of assembly demonstrations to learn from. The proposed procedure first disassembles

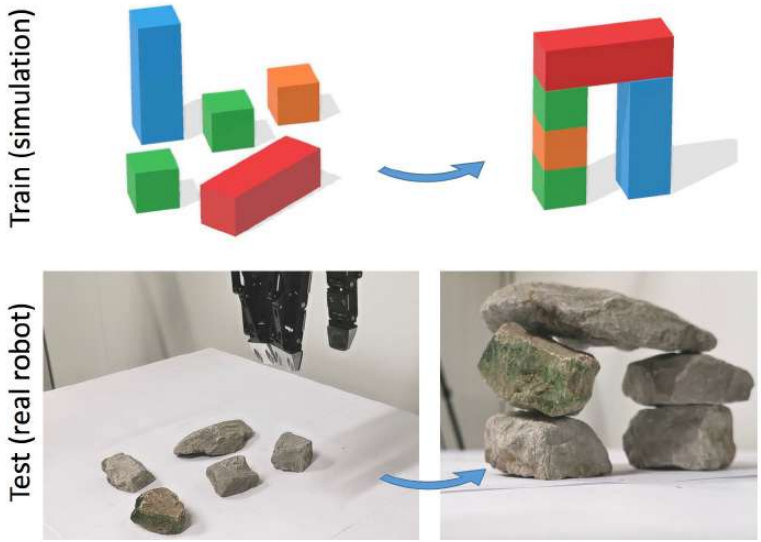


Figure 1.11 – Primitives on the left are assembled by learned policies into arch shapes on the right. The policies are trained on simulated data and transferred to the real world to assemble 3D objects from unseen primitives resembling building blocks used during training.

the given shape example and then reverses disassembly actions into an assembly demonstration. The control policies are trained using an automatically collected in simulation dataset of demonstrations and reach high success rates when evaluated on a real robot. We show that the learned visual policies generalize to assembling shapes using real-world building blocks never seen during training such as stones shown in the bottom of Figure 1.11. Moreover, we demonstrate that the learned policies can react to dynamic changes in real robot scenes. This work was published in IROS 2020 [Pashevich et al., 2020a] and is presented in Chapter 4.

- Our last contribution addresses the multimodal VLN problem where an agent navigates and interacts in a simulated environment given a goal defined with natural language. We propose Episodic Transformer (E.T.), an novel architecture based on the self-attention mechanism. In contrast to recurrent-based architectures relying on a hidden state, E.T. encodes all previous visual observations and actions history to be able to attend to objects seen before and actions taken in the past. Figure 1.12 shows an example of a compositional VLN task where actions which require attending to the past are highlighted with red arrows. To decouple understanding the visual ap-

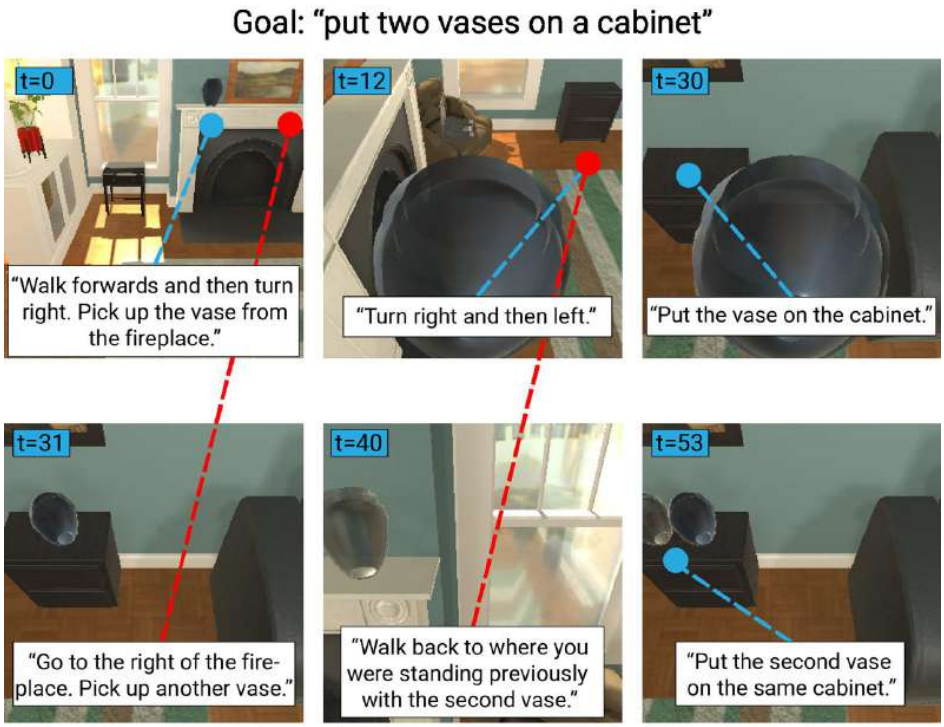


Figure 1.12 – An example of a compositional task in the ALFRED dataset [Shridhar et al., 2020] where the agent is asked to bring two vases to a cabinet. The figure shows six frames from an expert demonstration with corresponding step-by-step instructions. The instructions expect the agent to be able to navigate to a fireplace which is not visible in its current egocentric view and to remember its previous location by referring to it as "where you were standing previously".

pearance of an environment from the variations of natural language instructions, we propose to leverage synthetic instructions as an intermediate representation. We demonstrate that encoding longer history with self-attention is critical to accomplish compositional tasks and that pretraining and joint training with synthetic instructions further improve the performance. This work was published in ICCV 2021 [Pashevich et al., 2021a] and is presented in Chapter 5.

2

Learning to augment synthetic images for sim2real policy transfer

2.1 Introduction

Learning visuomotor control policies holds much potential for addressing complex robotics tasks in unstructured and dynamic environments. In particular, recent progress in computer vision and deep learning motivates new methods combining learning-based vision and control. Successful methods in computer vision share similar neural network architectures, but learn task-specific visual representations, e.g. for object detection, image segmentation or human pose estimation. Guided by this experience, one can assume that successful integration of vision and control will require learning of policy-specific visual representations for particular classes of robotics tasks.

Learning visual representations requires large amounts of training data. Previous work has addressed policy learning for simple tasks using real robots e.g., in [Levine et al., 2016, Pinto and Gupta, 2016, Zhang et al., 2018]. Given the large number of required attempts (e.g. 800,000 grasps collected in [Levine et al., 2016]), learning with real robots might be difficult to scale to more complex tasks and environments. On the other hand, physics simulators and graphics engines provide an attractive alternative due to the simple parallelization and scaling to multiple environments as well as due to access to the underlying world state during training.

Learning in simulators, however, comes at the cost of the reality gap. The difficulty of synthesizing realistic interactions and visual appearance typically induces biases and results in low performance of learned policies in real scenes. Among several approaches to address this problem, recent work proposes domain randomization [Tobin et al., 2017, Tobin et al., 2018]

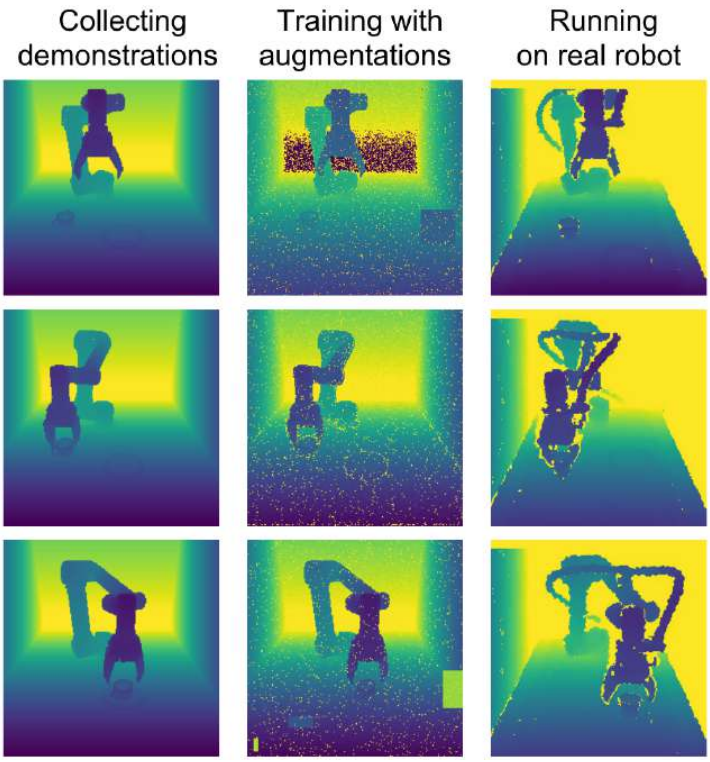


Figure 2.1 – Example depth images for the task "Cup placing". Synthetic depth maps (first column) are augmented with random transformations during policy training (second column). The resulting policy is applied to depth maps from the real robot scene (third column).

by augmenting synthetic data with random transformations such as random object shapes and textures. While demonstrating encouraging results for transferring simlator-trained policies to real environments ("sim2real" transfer), the optimality and generality of proposed transformations remains open.

In this work we follow the domain randomization approach and propose to learn transformations optimizing sim2real transfer. Given two domains, our method finds policy-independent sequences of random transformations that can be used to learn multiple tasks. While domain randomization can be applied to different stages of a simulator, our goal here is the efficient learning of visual representations for manipulation tasks. We therefore learn parameters of random transformations to bridge the domain gap between synthetic and real images. We here investigate the transfer of policies learned for depth images. However, our method should generalize to RGB inputs. Examples of our synthetic and real depth images used to train and

test the "Cup placing" policy are illustrated in Figure 2.1.

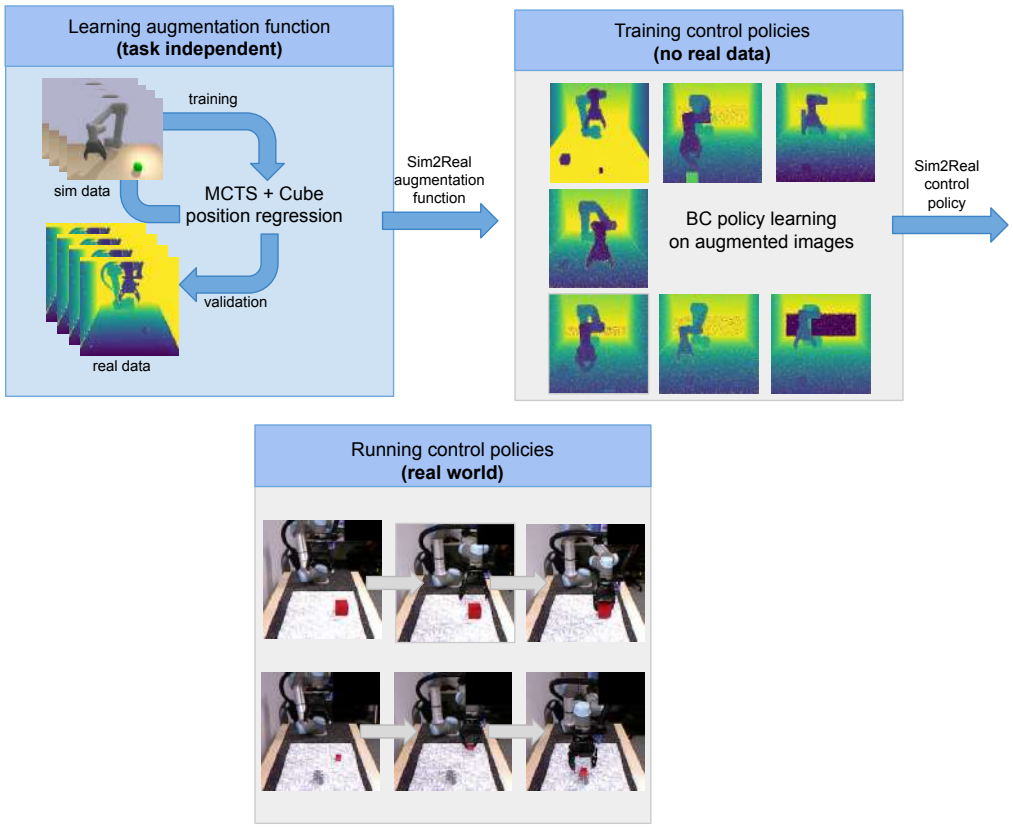


Figure 2.2 – Overview of the method. Our contribution is the policy-independent learning of depth image augmentations (top left). The resulting sequence of augmentations is applied to synthetic depth images while learning manipulation policies in a simulator (top right). The learned policies are directly applied to real robot scenes without finetuning on real images (bottom).

In more details, our method uses a proxy task of predicting object locations in a robot scene. We synthesize a large number of depth images with objects and train a CNN regressor estimating object locations after applying a given sequence of random transformations to synthetic images. We then score the parameters of current transformations by evaluating CNN location prediction on pre-recorded real images. Inspired by the recent success of AlphaGo [Silver et al., 2017] we adopt Monte-Carlo Tree Search (MCTS) [Coulom, 2006] as an efficient search strategy for transformation parameters.

We evaluate the optimized sequences of random transformations by applying them to simulator-based policy learning. We demonstrate the successful transfer of such policies to real robot scenes while using no real images for policy training. Our method is shown to generalize to multiple policies. The overview of our approach is illustrated in Figure 2.2. The code of the project is publicly available at the project website [Pashevich et al., 2019b].

2.2 Related work

Robotics tasks have been addressed by various learning methods including imitation learning [Duan et al., 2017] and reinforcement learning [Riedmiller et al., 2018]. Despite mastering complex simulated tasks such as Go [Silver et al., 2017] and Dota [OpenAI, 2018], the addressed robotics tasks remain rather simple [Zhang et al., 2018, Riedmiller et al., 2018, Gu et al., 2016]. This difference is mainly caused by the real-world training cost. Learning a manipulation task typically requires a large amount of data [Levine et al., 2016, Pinto and Gupta, 2016]. Thus, robot interaction time becomes much longer than in simulation [Gu et al., 2016] and expert guidance is non-trivial [Zhang et al., 2018].

Learning control policies in simulation and transferring them to the real world is a potential solution to address these difficulties. However, the visual input in simulation is significantly different from the real world and therefore requires adaptation [Sadeghi and Levine, 2017]. Recent attempts to bridge the gap between simulated and real images can be generally divided into two categories: domain adaptation [Bousmalis et al., 2018] and domain randomization [Tobin et al., 2017]. Domain adaptation methods either map both image spaces into a common one [James et al., 2018, Mueller et al., 2018] or map one into the other [Lee et al., 2018]. Domain randomization methods add noise to the synthetic images [Pinto et al., 2018, Sadeghi et al., 2018], thus making the control policy robust to different textures and lighting. The second line of work is attractive due to its effectiveness and simplicity. Yet, it was so far only shown to work with RGB images. While depth images are well suited for many robotics tasks [Litvak et al., 2019], it is not obvious what type of randomization should be used in the case of depth data. Here, we explore a learning based approach to select appropriate transformations and show that this allows us to close the gap between simulated and real visual data.

Domain randomization is also referred to as data augmentation in the context of image classification and object detection. Data augmentation

is known to be an important tool for training deep neural networks and in most cases it is based on a manually designed set of simple transformations such as mirroring, cropping and color perturbations. In general, designing an effective data augmentation pipeline requires domain-specific knowledge [Dvornik et al., 2018]. Depth images might be augmented by adding random noise [Handa et al., 2016], noise patterns typical for real sensors [Eitel et al., 2015] or by compensating missing information [Yang et al., 2012].

Learning to augment is a scalable and promising direction that has been explored for visual recognition in [Paulin et al., 2014]. Recent attempts to automatically find the best augmentation functions propose to use Reinforcement Learning and require several hundreds of GPUs [Cubuk et al., 2019]. Given the prohibitive cost of executing thousands of policies in a real-robot training loop, we propose to optimize sequences of augmentations within a proxy task by predicting object locations in pre-recorded real images using the Monte Carlo Tree Search [Coulom, 2006]. A related idea of learning rendering parameters of a simulator has been recently proposed for a different task of semantic image segmentation in [Ruiz et al., 2019].

2.3 Approach

We describe the proposed method for learning depth image augmentations in Sections 2.3.2 and 2.3.3. Our method builds on Behavior Cloning (BC) policy learning [Pomerleau, 1989, Ross and Bagnell, 2014] which we overview in Section 2.3.1.

2.3.1 Behavior cloning in simulation

Given a dataset $\mathcal{D}^{\text{expert}} = \{(o_t, a_t)\}$ of observation-action pairs along with the expert trajectories in *simulation*, we learn a function approximating the conditional distribution of the expert policy $\pi_{\text{expert}}(a_t|o_t)$ controlling a robotic arm. Here, the observation is a sequence of the three last depth frames, $o_t = (I_{t-2}, I_{t-1}, I_t) \in \mathcal{O} = \mathbb{R}^{H\times W\times 3}$. The action $a_t \in \mathcal{A} = \mathbb{R}^7$ is the robot command controlling the end-effector state. The action $a_t = (\boldsymbol{v}_t, \boldsymbol{\omega}_t, g_t)$ is composed of the end-effector linear velocity $\boldsymbol{v}_t \in \mathbb{R}^3$, end-effector angular velocity $\boldsymbol{\omega}_t \in \mathbb{R}^3$ and the gripper openness state $g_t \in \{0, 1\}$. We learn the deterministic control policy $\pi : \mathcal{O} \to \mathcal{A}$ approximating the expert policy π_{expert}. We define π by a Convolutional Neural Network (CNN) parameterized by a set of weights θ and learned by minimizing the L_2 loss

for the velocity controls $(\boldsymbol{v}_t, \boldsymbol{\omega}_t)$ and the cross-entropy loss L_{CE} for the binary grasping signal g_t. Given the state-action pair (s_t, a_t) and the network prediction $\pi_\theta(s_t) = (\hat{\boldsymbol{v}}_t, \hat{\boldsymbol{\omega}}_t, \hat{g}_t)$, we minimize the loss:

$$\lambda L_2\left([\hat{\boldsymbol{v}}_{\boldsymbol{t}}, \hat{\boldsymbol{\omega}}_t], [\boldsymbol{v}_t, \boldsymbol{\omega}_t]\right) + (1-\lambda) L_{\text{CE}}\left(\hat{g}_t, g_t\right), \tag{2.1}$$

where $\lambda \in [0, 1]$ is a scaling factor of the cross-entropy loss which we experimentally set to 0.9.

2.3.2 Sim2Real transfer

Given a stochastic augmentation function f, we train a CNN h to predict the cube position on a simulation dataset $\mathcal{D}^{\text{sim}} = \left\{(I_i^{\text{sim}}, p_i^{\text{sim}})\right\}$. Given an image I_i^{sim}, the function f sequentially applies N primitive transformations, each with a certain probability. This allows for bigger variability during the training. We minimize the L_2 loss between the cube position p_i^{sim} and the network prediction given an augmented depth image $h\left(f(I_i^{\text{sim}})\right)$:

$$\sum_k \mathbb{E}\, L_2\left(h\left(f(I_k^{\text{sim}})\right), p_k^{\text{sim}}\right). \tag{2.2}$$

We evaluate augmentation functions by computing the average error of network prediction on a pre-recorded real-world dataset, $\mathcal{D}^{\text{real}} = \left\{(I_i^{\text{real}}, p_i^{\text{real}})\right\}$ as

$$e^{real} = \frac{1}{n} \sum_{k=1}^{n} L_2\left(h(I_k^{\text{real}}), p_k^{\text{real}}\right). \tag{2.3}$$

The optimal augmentation function f^* should result in a network h with the smallest real-world error e^{real}. We assume that the same augmentation function will produce optimal control policies. We re-apply the learned stochastic function f^* on individual frames of $\mathcal{D}^{\text{expert}}$ at every training epoch and learn π^{sim2real}. We use π^{sim2real} to control the real robot using the same control actions as in the simulation, i.e., $a_t^{\text{real}} = (\boldsymbol{v}_t^{\text{real}}, \boldsymbol{\omega}_t^{\text{real}}, g_t^{\text{real}}) = \pi^{\text{sim2real}}(I_t^{\text{real}})$, see Figure 2.2.

2.3.3 Augmentation space

We discretize the search space of augmentation functions by considering sequences of N transformations from a predefined set. We then apply the selected sequence of transformations in a given order to each image. The predefined set of transformations consists of the depth-applicable standard transformations from PIL, a popular Python Image Library [Clark, 2015], as well as Cutout [DeVries and Taylor, 2017], white (uniform) noise and salt

(bernoulli) noise. We also take advantage of segmentation masks provided by the simulator and define two object-aware transformations, i.e., boundary noise and object erasing (see Section 2.4.2 for details). The identity (void) transformation is included in the set to enable the possibility of reducing N. The full set of our eleven transformations is listed in Table 2.1. Each transformation is associated with a magnitude and a probability of its activation. The magnitude defines the transformation-specific parameter. For each transformation we define two possible magnitudes and three probabilities. With $N = 8$ in our experiments, our search space roughly includes $(11 \times 2 \times 3)^8 \approx 3.6 * 10^{14}$ augmentation functions. We reduce the search space by restricting each transformation, except identity, to occur only once in any augmentation sequence.

2.3.4 Real robot control

Algorithm 1 Sim2Real policy transfer algorithm

1: *** Given datasets $\mathcal{D}^{\text{sim}}, \mathcal{D}^{\text{real}}, \mathcal{D}^{\text{expert}}_{\text{sim}}$ ***
2: MCTS = init_mcts()
3: **repeat**
4: f = MCTS.sample_path()
5: CNN = train_cube_prediction($f, \mathcal{D}^{\text{sim}}$) ▷ Equation 2.2
6: e^{real} = compute_error(CNN, $\mathcal{D}^{\text{real}}$) ▷ Equation 2.3
7: MCTS.update(e^{real}) ▷ Backpropagate the error
8: **until** the smallest e^{real} is constant for 500 iterations
9: f^* = MCTS.select_best_path()
10: π^{sim2real} = train_BC_policy($f^*, \mathcal{D}^{\text{expert}}_{\text{sim}}$) ▷ Equation 2.1
11: **return** π^{sim2real}

To find an optimal sequence of augmentations, we use Monte Carlo Tree Search (MCTS) [Coulom, 2006] which is a heuristic tree search algorithm with a trade-off between exploration and exploitation. Our search procedure is defined in Algorithm 1. The algorithm iteratively explores the Monte Carlo tree (lines 3-8) by sampling sequences of transformations (line 4), training a cube position prediction network on an augmented simulation dataset (line 5), evaluating the trained network on the real dataset (line 6) and backpropagating the evaluation error through the Monte Carlo tree (line 7). Once the smallest error on the real dataset stays constant for 500 iterations, we choose the best augmentation function according to MCTS (line 9) and train sim2real control policies using the simulation dataset of augmented expert trajectories on the tasks of interest (line 10). Once

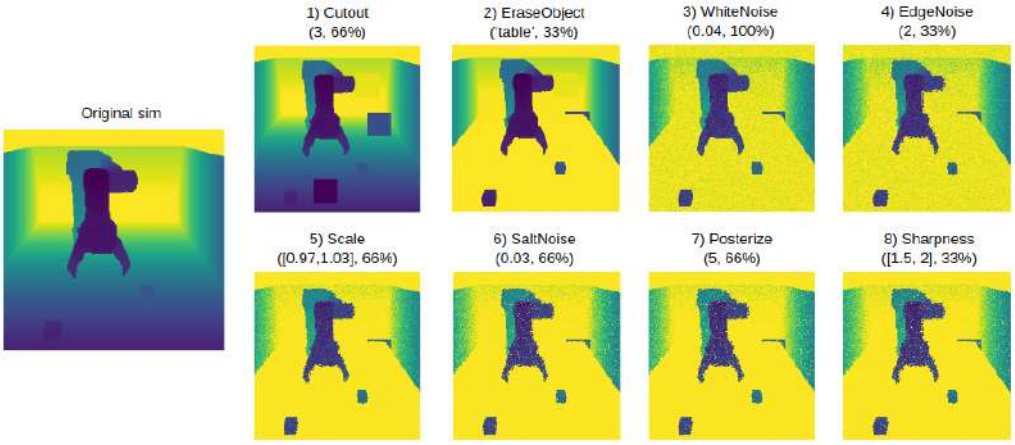


Figure 2.3 – The original synthetic depth image on the left is augmented by the sequence of eight random transformations learned by our method.

trained, the control policies can be directly applied in real robot scenes without finetuning. The sequence of eight transformations found by MCTS is illustrated in Figure 2.3.

2.4 Results

This section evaluates the transfer of robot control policies from simulation to real. First, we describe our tasks and the experimental setup in Section 2.4.1. We evaluate independently each of predefined basic transformations on the cube position prediction task in Section 2.4.2. In Section 2.4.3, we compare our approach of learning augmentation functions with baselines. Finally, we demonstrate the policy transfer to the real-world robotics tasks in Section 2.4.4.

2.4.1 Experimental setup

Our goal is to learn a policy to control a UR5 6-DoF robotic arm with a 3 finger Robotiq gripper for solving manipulation tasks in the real world. The policy takes as input 3 depth images $o_t \in \mathbb{R}^{H \times W \times 3}$ from the Kinect-1 camera positioned in front of the arm. We scale the values of depth images to the range $[0, 1]$. The policy controls the robot with an action $a_t \in \mathbb{R}^7$. The control is performed at a frequency of 10 Hz. All the objects and the robotic gripper end-effector are initially allocated within the area of $60 \times 60\,\mathrm{cm}^2$ in front of the arm. The simulation environment is built with

the `pybullet` physics simulator [Coumans, 2009] and imitates the real-world setup.

We consider three manipulation tasks.

- Cube picking task shown in Figure 2.5a. The goal of the task is to pick up a cube of size 4.7 cm and to lift it. In simulation, the cube size is randomized between 3 and 9 cm.
- Cubes stacking task shown in Figure 2.5b. The goal of the task is to stack a cube of size 3.5 cm on top of a cube of size 4.7 cm. We randomize the sizes of cubes in simulation between 3 to 9 cm.
- Cup placing task shown in Figure 2.6. The goal of the task is to pick up a cup and to place it on a plate. In simulation, we randomly sample 43 plates from ModelNet [Wu et al., 2015] and 134 cups from ShapeNet [Chang et al., 2015]. We use three cups and three plates of different shapes in our real robot experiments.

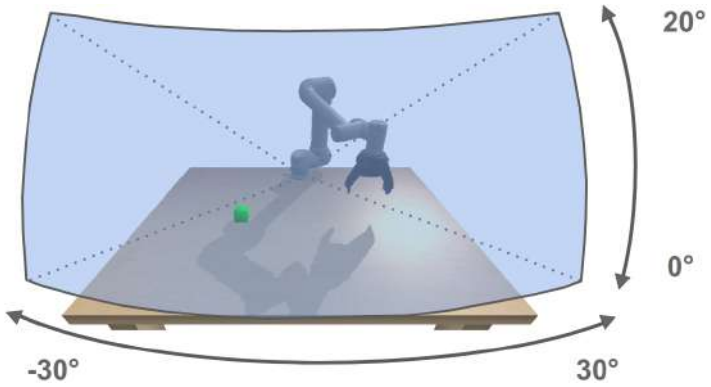


Figure 2.4 – For demonstrations recorded in simulation, we randomly sample several viewpoints in a section of a sphere centered around a reference viewpoint.

2.4.2 Evaluation of individual transformations

Before learning complex augmentation functions, we independently evaluate each transformation from the set of transformations defined in Section 2.3.3. We first describe each transformation and the associated values of magnitude:

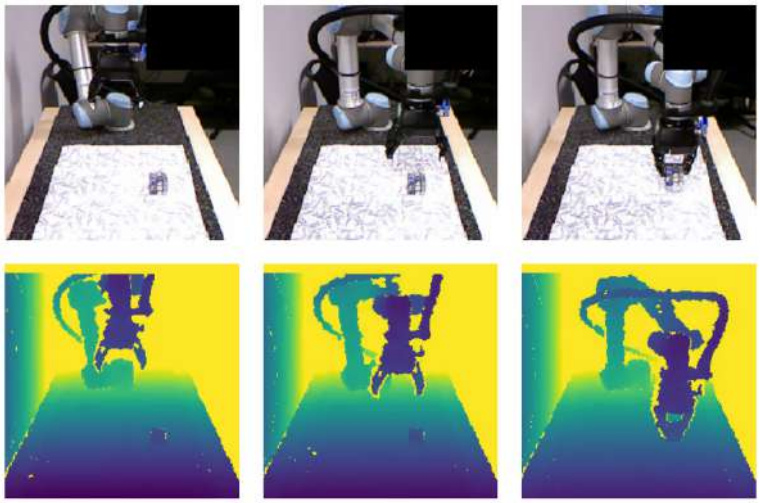
- Affine transformation randomly translates and rotates the image in the range of $[-9, 9]$ pixels and $[-5, 5]^\circ$, or alternatively by $[-16, 16]$ pixels and $[-10, 10]^\circ$.

Transformation	Error in sim	Error in real
Identity	0.63 ± 0.50	6.52 ± 5.04
Affine	$\mathbf{0.59 \pm 0.45}$	4.83 ± 4.42
Cutout	1.19 ± 0.87	$\mathbf{1.86 \pm 2.45}$
Invert	0.88 ± 0.60	4.63 ± 3.28
Posterize	0.66 ± 0.48	5.54 ± 4.59
Scale	0.67 ± 0.47	6.00 ± 4.37
Sharpness	0.83 ± 0.49	5.48 ± 3.84
WhiteNoise	0.68 ± 0.54	3.60 ± 2.33
SaltNoise	0.72 ± 0.50	2.42 ± 1.16
BoundaryNoise	0.88 ± 0.66	2.06 ± 1.17
EraseObject	0.64 ± 0.47	1.93 ± 1.01

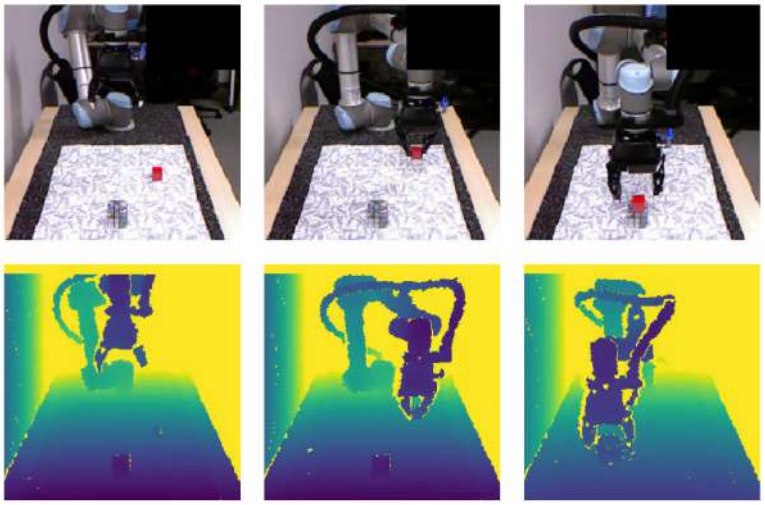
Table 2.1 – Cube prediction error (in cm) for synthetic and real depth images evaluated separately for each of eleven transformations considered in this work. The errors are averaged over 200 pairs of images and cube positions.

- Cutout [DeVries and Taylor, 2017] samples one or three random rectangles in the image and sets their values to a random constant in $[0, 1]$.
- Invert function inverts each pixel value by applying the operation $x \mapsto 1 - x$ and does not have any parameters.
- Posterize transformation reduces the number of bits for each pixel value to be either 5 or 7.
- Scale transformation randomly multiplies the image with a constant in one of the two ranges: $[0.95, 1.05]$ or $[0.97, 1.03]$.
- Sharpness transformation increases the image sharpness either randomly in the range between 50% and 100% or by 100%.
- WhiteNoise transformation adds uniform noise to each pixel with a magnitude of 0.04 or 0.08.
- SaltNoise transformation sets each pixel value to 1 with the probability 0.01 or 0.03.
- BoundaryNoise transformation uses the semantics mask of the simulator and removes patches of pixels located at the boundary between different objects. For BoundaryNoise, we remove either 2 or 4 pixels along the boundaries.
- EraseObject transformation removes either the table or the walls behind the robot using the semantics segmentation mask.

All the above transformations are associated with a probability in the set



(a) Cube picking task

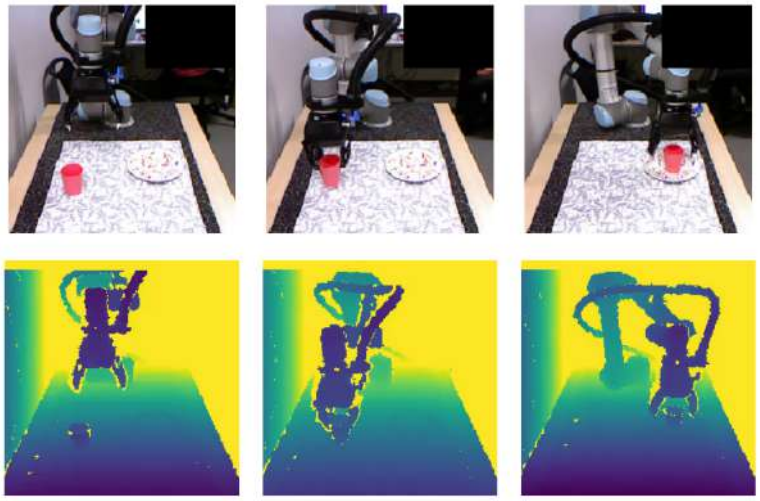


(b) Cube stacking task

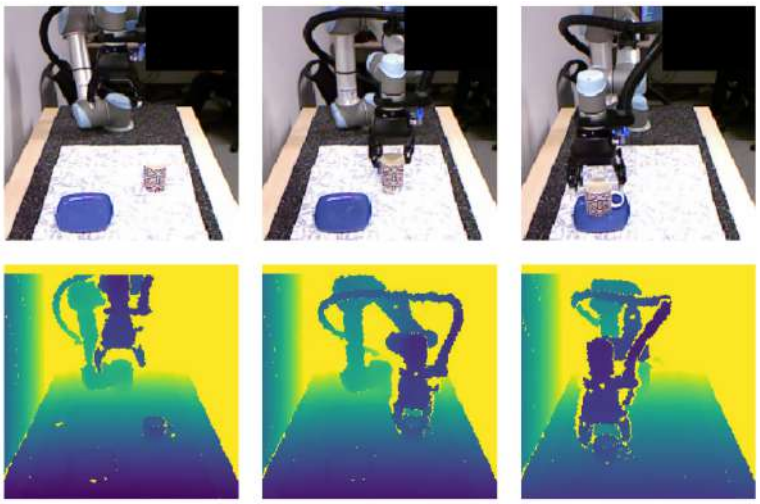
Figure 2.5 – Frame sequences (RGB and depth) of real-world tasks performed with a policy learned on a simulation dataset augmented with our approach. The tasks are: a) picking up a cube, b) stacking two cubes.

$\{33\%, 66\%, 100\%\}$. For instance, the probability 33% means that the transformation is applied 1 out of 3 times and 2 our of 3 times it acts as the identity function.

As explained in Section 2.3.2, we evaluate each augmentation function by computing the prediction error of the cube position in real depth images after training the position regressor on simulated data. We collect 2000



(a) Cup placing task



(b) Cup placing task with another set of objects

Figure 2.6 – Frame sequences (RGB and depth) of real-world tasks performed with a policy learned on a simulation dataset augmented with our approach. Both figures illustrate the cup placing task where different sets of objects are used.

pairs of simulated depth images and cube positions for training and 200 real depth images for evaluation. To be robust to viewpoint changes in real scenes, each simulated scene is recorded from five random viewpoints. We randomize the camera viewpoint in simulation around the frontal viewpoint by sampling the camera yaw angle in $[-15, 15]^\circ$, pitch angle in $[15, 30]^\circ$,

and distance to the robot base in $[1.35, 1.50]$ m as shown in Figure 2.4. We only require the viewpoint of the real dataset to be within the simulated viewpoints distribution. Moreover, we allow to move the camera between different real robot experiments.

We treat each transformation in the given set as a separate augmentation function (sequence of length 1) and use each of them independently to augment the simulation dataset. Next, we train a CNN based on ResNet-18 architecture [He et al., 2016] to predict the cube position given a depth image. During the network training, we compute the prediction error with Equation 2.3 on two validation datasets, 200 simulated images and 200 real images. To evaluate each augmentation function more robustly, we always start the CNN training at the same initial position. For each transformation, we report the cube position prediction error in Table 2.1. With no data augmentation, the trained network performs well in simulation (error of 0.63 cm) but works poorly on real images (error of 6.52 cm). Cutout transformation reduces the regression error by more than 4 cm and indicates that it should be potentially combined with other transformations.

2.4.3 Augmentation function learning

We compare the augmentation function learned by our approach to several baselines on the task of estimating the cube position in Table 2.2. The baselines include training the network on synthetic images (i) without any data augmentation, (ii) with augmentation sequences composed of 8 random transformations (average over 10 random sequences), (iii) with a handcrafted augmentation sequence of four transformations built according to our initial intuition: Scale, WhiteNoise, EraseObjects and SaltNoise, (iv) with the best single transformation from Section 2.4.2 and (v) with the learned augmentation composed of 4 transformations. To have a robust score and exclude outliers, we compute the median error over evaluations of 10 training epochs.

Baselines (i)-(iii) with no augmentation learning demonstrate worst results on the real dataset. Results for learned transformations (iv)-(vi) show that more transformations with different probabilities help to improve the domain transfer. Table 2.2 also demonstrates the trade-off between the performance in different domains: the better augmentation works on the real dataset, the worse it performs in the simulation. Effectively, the learned augmentation shifts the distribution of simulated images towards the distribution of real images. As a consequence, the network performs well on the real images that are close to the training set distribution and works

	Augmentation	Error in sim	Error in real
i	None	$\mathbf{0.63 \pm 0.50}$	6.52 ± 5.04
ii	Random (8 operations)	6.56 ± 4.05	5.77 ± 3.12
iii	Handcrafted (4 operations)	0.99 ± 0.68	2.35 ± 1.36
iv	Learned (1 operation)	1.19 ± 0.87	1.86 ± 2.45
v	Learned (4 operations)	1.21 ± 0.78	1.17 ± 0.71
vi	Learned (8 operations)	1.31 ± 0.90	$\mathbf{1.09 \pm 0.73}$

Table 2.2 – Cube prediction error (in cm) on synthetic and real depth images using different types of depth data augmentation. More augmentations increase the error for synthetic images and decrease the error for real images as expected from our optimization procedure. The errors are averaged over 200 pairs of images and cube positions.

Augmentation	Pick	Stack	Cup Placing
None	3/20	1/20	0/20
Handcrafted (4 operations)	9/20	2/20	6/20
Learned (1 operation)	8/20	1/20	1/20
Learned (8 operations)	**19/20**	**18/20**	**15/20**

Table 2.3 – Success rates for control policies executed on a real robot (20 trials per experiment). Results are shown for three tasks and alternative depth image augmentations.

worse on the original simulated images that lie outside of the training set distribution.

The best augmentation sequence found by our method is illustrated in Figure 2.3 and contains the following transformations: Cutout, EraseObject, WhiteNoise, EdgeNoise, Scale, SaltNoise, Posterize, Sharpness. Learning an augmentation function of length 8 takes approximately 12 hours on 16 GPUs. The vast majority of this time is used to train the position estimation network while MCTS path sampling, evaluation and MCTS backpropagation are computationally cheap. We iteratively repeat the training and evaluation routine until the error does not decrease for a sufficiently long time (500 iterations). Once the sim2real augmentation is found, it takes approximately an hour to train the BC control policy.

2.4.4 Real robot control

In this section we demonstrate that the data augmentation learned for a proxy task transfers to other robotic control tasks. We collect expert demonstrations where the full state of the system is known and an expert script can easily be generated at training time. We augment the simulated demonstrations with the learned data augmentation and train BC policies without any real images. Moreover, we show that our augmentation is not object specific and transfers to tasks with new object instances not present in the set of expert demonstrations. For each task, we compare the learned augmentation function with 3 baselines: no augmentation, handcrafted augmentation and best single transformation. Each evaluation consists of 20 trials with random initial configurations. The results are reported in Table 2.3.

Cube picking task. Success rates for policies learned with different augmentation functions are strongly correlated with results for cube position estimation in Table 2.2. The policy without augmentation has a success rate 3/20. Single transformation and handcrafted augmentation have 9/20 and 8/20 successful trials respectively. The sim2real policy learned with our method succeeds 19 out of 20 times.

Cube stacking task. Given a more difficult task where more precision is required, the baseline approaches perform poorly and achieve the success rate of only 2/20 for the handcrafted augmentation. We observe most of the failure cases due to imprecise grasping and stacking. We successfully tested the learned data augmentation function on cubes of varying sizes which indicates high control precision. Overall, our method was able to stack cubes in 18 runs out of 20.

Cup placing. Solving the Cup placing task requires both precision and the generalization to previously unseen object instances. The policies are trained over a distribution of 3D meshes and thus leverage the large dataset available in the simulation. All baselines fail to solve the Cup placing except for the handcrafted augmentation which succeeds 6 times out of 20. Our approach is able to solve the task with the success rate of 15/20 despite the presence of three different instances of cups and plates never seen during training. These results confirm our hypothesis that the augmentations learned for a proxy task of predicting the cube position, generalize to new objects and tasks.

2.5 Conclusion

In this chapter, we introduce a method to learn augmentation functions for sim2real policy transfer. The learned augmentation function consists of a sequence of random transformations such as scaling and adding noise and is used to augment synthetic images. Once augmented, synthetic images are used for policy learning using the behaviour cloning approach. The control policies learned on augmented synthetic images can be applied directly on a real robot without any further finetuning. To evaluate the transfer, we propose a proxy task of object position estimation that requires only a small amount of real-world data. Our evaluation of data augmentation shows significant improvement over the baselines. We also show that the performance on the proxy task strongly correlates with the final policy success rate. Our method does not require any real images for policy learning and can be applied to various manipulation tasks. We apply our approach to solve three real-world tasks including the task of manipulating previously unseen objects.

3

Learning to combine primitive skills: A step towards versatile robotic manipulation

3.1 Introduction

In this chapter, we consider visually guided robotics manipulations and aim to learn robust visuomotor control policies for particular tasks. Autonomous manipulations such as assembling IKEA furniture [Suárez-Ruiz et al., 2018] remain highly challenging given the complexity of real environments as well as partial and uncertain observations provided by the sensors. Successful methods for task and motion planning (TAMP) [Srivastava et al., 2014, Lozano-Pérez and Kaelbling, 2014, Toussaint, 2015] achieve impressive results for complex tasks but often rely on limiting assumptions such as the full state observability and known 3D shape models for manipulated objects. Moreover, TAMP methods usually complete planning before execution and are not robust to dynamic scene changes.

Recent learning methods aim to learn visuomotor control policies directly from image inputs. Imitation learning (IL) [Pomerleau, 1989, Ross and Bagnell, 2014, Pinto et al., 2018, Ng and Russell, 2000] is a supervised approach that can be used to learn simple skills from expert demonstrations. One drawback of IL is its difficulty to handle new states that have not been observed during demonstrations. While increasing the number of demonstrations helps to alleviate this issue, an exhaustive sampling of action sequences and scenarios becomes impractical for long and complex tasks.

In contrast, reinforcement learning (RL) requires little supervision and achieves excellent results for some challenging tasks [Mnih et al., 2015, Sil-

ver et al., 2016]. RL explores previously unseen scenarios and, hence, can generalize beyond expert demonstrations. As full exploration is exponentially hard and becomes impractical for problems with long horizons, RL often relies on careful engineering of rewards designed for specific tasks.

Common tasks such as preparing food or assembling furniture require long sequences of steps composed of many different actions. Such tasks have long horizons and, hence, are difficult to solve by either RL or IL methods alone. To address this issue, we propose a RL-based method that learns to combine simple imitation-based policies. Our approach simplifies RL by reducing its exploration to sequences with a limited number of primitive actions, that we call skills.

Given a set of pretrained skills such as "grasp a cube" or "pour from a cup", we train RL with sparse binary rewards corresponding to the correct/incorrect execution of the full task. While hierarchical policies have been proposed in the past [Das et al., 2018, Le et al., 2018], our approach can learn composite manipulations using no intermediate rewards and no demonstrations of full tasks. Hence, the proposed method can be directly applied to learn new tasks. See Figure 3.1 and Figure 3.2 for an overview of our approach.

Our skills are low-level visuomotor controllers learned from synthetic demonstrated trajectories with behavioral cloning (BC) [Pomerleau, 1989]. Examples of skills include go to the bowl, grasp the object, pour from the held object, release the held object, etc. We automatically generate expert synthetic demonstrations and learn corresponding skills in simulated environments. We also minimize the number of required demonstrations by choosing appropriate CNN architectures and data augmentation methods. Our approach is shown to compare favorably to the state of the art [Pinto et al., 2018] on the FetchPickPlace test environment [Plappert et al., 2018]. Moreover, using the sim2real approach from the previous chapter [Pashevich et al., 2019a] we demonstrate the successful transfer and high accuracy of our simulator-trained policies when tested on a real robot.

We compare our approach with two classical methods: (a) an open-loop controller estimating object positions and applying a standard motion planner (b) a closed-loop controller adapting the control to re-estimated object positions. We show the robustness of our approach to a variety of perturbations. The perturbations include dynamic change of object positions, new object instances and temporary object occlusions. The versatility of learned policies comes from both the reactivity of the BC learned skills and the ability of the RL master policy to re-plan in case of failure. Our approach allows to compute adaptive control and planning in real-time.

In summary, this chapter makes the following contributions. (i) We pro-

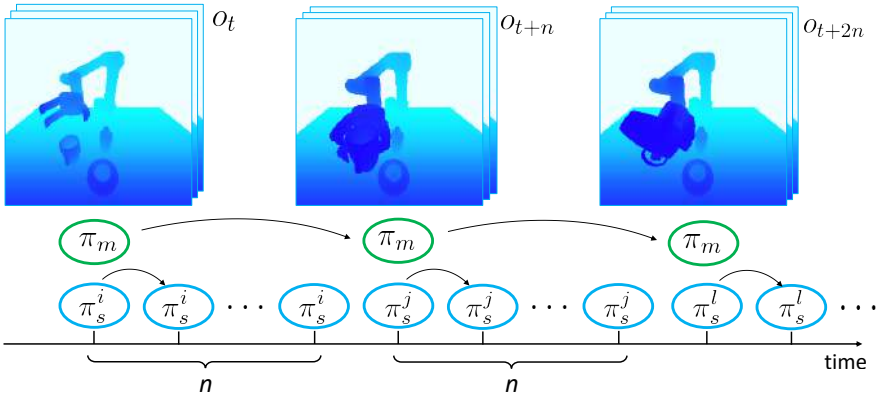


Figure 3.1 – Temporal hierarchy of master and skill policies. The master policy π_m is executed at a coarse interval of n time-steps to select among K skill policies $\pi_s^1 \dots \pi_s^K$. Each skill policy generates control for a primitive action such as grasping or pouring.

pose to learn robust RL policies that combine BC skills to solve composite tasks. (ii) We present sample efficient training of BC skills and demonstrate an improvement compared to the state of the art. (iii) We demonstrate successful learning of relatively complex manipulation tasks with neither intermediate rewards nor full demonstrations. (iv) We successfully transfer and execute policies learned in simulation to real robot setups. (v) We show successful task completion in the presence of perturbations. Our simulation environments and the code used in this work are publicly available on our website [Strudel et al., 2020a].

3.2 Related work

Our work is related to robotics manipulation such as grasping [Lampe and Riedmiller, 2013], opening doors [Gu et al., 2016], screwing the cap of a bottle [Levine et al., 2015] and cube stacking [Popov et al., 2017]. Such tasks have been addressed by various methods including imitation learning (IL) [Duan et al., 2017] and reinforcement learning (RL) [Riedmiller et al., 2018].

Imitation learning (IL). A neural network is trained to solve a task by observing demonstrations. Approaches include behavioral cloning (BC) [Pomer-

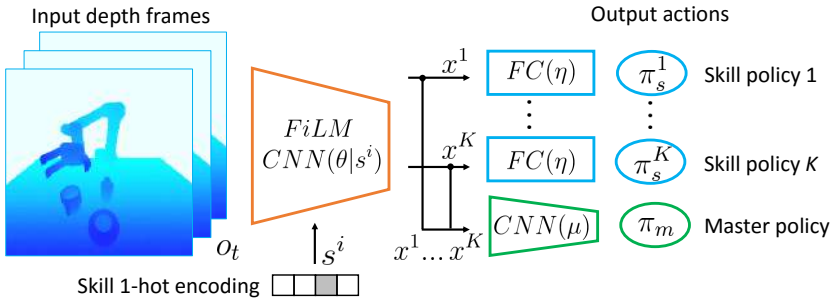


Figure 3.2 – CNN architecture used for the skill and master policies. The network is conditioned on one-hot skill encoding using FiLM encoding and outputs master choice and skill actions simultaneously.

leau, 1989] and inverse reinforcement learning [Ng and Russell, 2000]. BC learns a function that maps states to expert actions [Ross and Bagnell, 2014, Pinto et al., 2018], whereas inverse reinforcement learning learns a reward function from demonstrations in order to solve the task with RL [Ho and Ermon, 2016, Kumar et al., 2016, Popov et al., 2017]. BC typically requires a large number of demonstrations and has issues with not observed trajectories. While these problems might be solved with additional expert supervision [Ross and Bagnell, 2014] or noise injection in expert demonstrations [Laskey et al., 2017], we address them by improving the standard BC framework. We use recent state-of-the-art CNN architectures and data augmentation for expert trajectories. This permits to significantly reduce the number of required demonstrations and to improve performance.

Reinforcement learning (RL). RL learns to solve a tasks without demonstrations using exploration. Despite impressive results in several domains [Silver et al., 2016, Mnih et al., 2015, Kober et al., 2013, Gu et al., 2016], RL methods show limited capabilities when operating in complex and sparse-reward environments common in robotics. Moreover, RL methods typically require prohibitively large amounts of interactions with the environment during training. Hierarchical RL (HRL) methods alleviate some of these problems by learning a high-level policy modulating low-level workers. HRL approaches are generally based either on options [Sutton et al.,

1999] or a feudal framework [Dayan and Hinton, 1993]. The option methods learn a master policy that switches between separate skill policies [Frans et al., 2018, Lee et al., 2019, Bacon et al., 2017, Florensa et al., 2017a]. The feudal approaches learn a master policy that modulates a low-level policy by a control signal [Haarnoja et al., 2018, Nachum et al., 2018, Vezhnevets et al., 2017, Kulkarni et al., 2016, Hausman et al., 2018]. Our approach is based on options but in contrast to the cited methods, we pretrain the skills with IL. This allows us to solve complex and sparse reward problems using significantly less interactions with the environment during training.

Combining RL and IL. A number of approaches combining RL and IL have been introduced recently. Gao et al. [Gao et al., 2018] use demonstrations to initialize the RL agent. In [Cheng et al., 2018, Sun et al., 2018] RL is used to improve expert demonstrations, but does not learn hierarchical policies. Demonstrations have also been used to define RL objective functions [Hester et al., 2018, Nair et al., 2018] and rewards [Zhu et al., 2018]. Das et al. [Das et al., 2018] combine IL and RL to learn a hierarchical policy. Unlike our method, however, [Das et al., 2018] requires full task demonstrations and task-specific reward engineering. Moreover, the addressed navigation problem in [Das et al., 2018] has a much lower time horizon compared to our tasks. [Das et al., 2018] also relies on pretrained CNN representations which limits its application domain. Le at al. [Le et al., 2018] train low-level skills with RL, while using demonstrations to switch between skills. In a reverse manner, we use IL to learn low-level control and then deploy RL to find appropriate sequences of pretrained skills. The advantage is that our method can learn a variety of complex manipulations without full task demonstrations. Moreover, [Das et al., 2018, Le et al., 2018] learn discrete actions and cannot be directly applied to robotics manipulations that require continous control.

In summary, none of the methods [Das et al., 2018, Le et al., 2018, Cheng et al., 2018, Sun et al., 2018] is directly suitable for learning complex robotic manipulations due to requirements of dense rewards [Das et al., 2018, Sun et al., 2018] and state inputs [Cheng et al., 2018, Sun et al., 2018], limitations to short horizons and discrete actions [Das et al., 2018, Le et al., 2018], the requirement of full task demonstrations [Das et al., 2018, Le et al., 2018, Cheng et al., 2018, Sun et al., 2018] and the lack of learning of visual representations [Das et al., 2018, Cheng et al., 2018, Sun et al., 2018]. Moreover, our skills learned from synthetic demonstrated trajectories outperform RL based methods, see Section 3.5.4.

3.3 Approach

Our RLBC approach aims to learn multi-step policies by combining reinforcement learning (RL) and pretrained skills obtained with behavioral cloning (BC). We present BC and RLBC in Sections 3.3.1 and 3.3.2. Implementation details are given in Section 3.3.3.

3.3.1 Skill learning with behavioral cloning

Our first goal is to learn basic skills that can be composed into more complex policies. Given observation-action pairs $\mathcal{D} = \{(o_t, a_t)\}$ along expert trajectories, we follow the behavioral cloning approach [Pomerleau, 1989] and learn a function approximating the conditional distribution of the expert policy $\pi_{\mathrm{E}}(a_t|o_t)$ controlling a robot arm. Our observations $o_t \in \mathcal{O} = \mathbb{R}^{\mathrm{H}\times\mathrm{W}\times\mathrm{M}}$ are sequences of the last M depth frames. Actions $a_t = (\mathbf{v}_t, \boldsymbol{\omega}_t, g_t)$, $a_t \in \mathcal{A}^{\mathrm{BC}}$ are defined by the end-effector linear velocity $\mathbf{v}_t \in \mathbb{R}^3$ and angular velocity $\boldsymbol{\omega}_t \in \mathbb{R}^3$ as well as the gripper openness state $g_t \in \{0, 1\}$.

We learn the deterministic skill policies $\pi_s : \mathcal{O} \to \mathcal{A}^{\mathrm{BC}}$ approximating the expert policy π_{E}. Given observations o_t with corresponding expert (ground truth) actions $a_t = (\mathbf{v}_t, \boldsymbol{\omega}_t, g_t)$, we represent π_s with a convolutional neural network (CNN) and learn network parameters (θ, η) such that predicted actions $\pi_s(o_t) = (\hat{\mathbf{v}}_t, \hat{\boldsymbol{\omega}}_t, \hat{g}_t)$ minimize the loss L_{BC}:

$$\lambda \left\| [\hat{\mathbf{v}}_\mathbf{t}, \hat{\boldsymbol{\omega}}_t] - [\mathbf{v}_t, \boldsymbol{\omega}_t] \right\|_2^2 + (1 - \lambda) \left(g_t \log \hat{g}_t + (1 - g_t) \log \left(1 - \hat{g}_t\right)\right), \quad (3.1)$$

where $\lambda \in [0, 1]$ is a scaling factor which we empirically set to 0.9.

Our network architecture is presented in Figure 3.2. When training a skill policy π_s^i, such as reaching, grasping or pouring, we condition the network on the skill using the recent FiLM architecture [Perez et al., 2018]. Given the one-hot encoding s^i of a skill i, we use s^i as input to the FiLM generator which performs affine transformations of the network feature maps. FiLM conditions the network on performing a given skill, which permits learning a shared representation for all skills. Given an observation o_t, the network $CNN(\theta|s^i)$ generates a feature map x_t^i conditioned on skill i. The spatially-averaged x_i is linearly mapped with $FC(\eta)$ to the action of π_s^i.

3.3.2 RLBC approach

We wish to solve composite manipulations without full expert demonstrations and with a single sparse reward. For this purpose we rely on a high-level *master policy* π_m controlling the pretrained *skill policies* π_s at a coarse timescale. To learn π_m, we follow the standard formulation of

reinforcement learning and maximize the expected return $\mathbb{E}_\pi \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ given rewards r_t. Our reward function is sparse and returns 1 upon successful termination of the task and 0 otherwise. The RL master policy $\pi_m : \mathcal{O} \times \mathcal{A}^{\text{RL}} \rightarrow [0, 1]$ chooses one of the K skill policies to execute the low-level control, i.e., the action space of π_m is discrete: $\mathcal{A}^{\text{RL}} = \{1, \ldots, K\}$. Note, that our sparse reward function makes the learning of deep visual representations challenging. We, therefore, train π_m using visual features x_t^i obtained from the BC pretrained $CNN(\theta|s^i)$. Given an observation o_t, we use the concatenation of skill-conditioned features $\{x_t^1, \ldots, x_t^K\}$ as input for the master $CNN(\mu)$, see Figure 3.2.

To solve composite tasks with sparse rewards, we use a coarse timescale for the master policy. The selected skill policy controls the robot for n consecutive time-steps before the master policy is activated again to choose a new skill. This allows the master to focus on high-level task planning rather than low-level motion planning achieved by the skills. We expect the master policy to recover from unexpected events, for example, if an object slips out of a gripper, by re-activating an appropriate skill policy. Our combination of the master and skill policies is illustrated in Figure 3.1.

RLBC algorithm. The pseudo-code for the proposed approach is shown in Algorithm 2. The algorithm can be divided into three main steps. First, we collect a dataset of expert trajectories $\mathcal{D}_k$ for each skill policy π_s^k. For each policy, we use an expert script that has an access to the full state of the environment. Next, we train a set of skill policies $\{\pi_s^1, \ldots, \pi_s^K\}$. We sample a batch of state-action pairs and update parameters of convolutional layers θ and the skills linear layer parameters η. Finally, we learn the master π_m using the pretrained skill policies and the frozen parameters θ. We collect episode rollouts by first choosing a skill policy with the master and then applying the selected skill to the environment for n time-steps. We update the master policy weights μ to maximize the expected sum of rewards.

3.3.3 Approach details

Skill learning with BC. We use ResNet-18 for the $CNN(\theta|s^i)$, which we compare to VGG16 and ResNet-101 in Section 3.5.1. We augment input depth frames with random translations, rotations and crops. We also perform viewpoint augmentation and sample the camera positions on a section of a sphere centered on the robot and with a radius of 1.40 m. We uniformly sample the yaw angle in $[-15°, 15°]$, the pitch angle in $[15°, 30°]$, and the distance to the robot base in $[1.35, 1.50]$ m. The impact of both augmentations is evaluated in Section 3.5.2. We normalize the ground truth

Algorithm 2 RLBC approach algorithm

*** Collect expert data ***
for $k \in \{1, \ldots, K\}$ **do**
    Collect an expert dataset $\mathcal{D}_k$ for the skill policy π_s^k
*** Train $\{\pi_s^1, \ldots, \pi_s^K\}$ by solving: ***
$\theta, \eta = \arg\min_{\theta,\eta} \sum_{k=1}^{K} \sum_{(o_t,a_t)\in\mathcal{D}_k} L_{\text{BC}}(\pi_s^k(o_t), a_t)$ ▷ Equation 3.1
while task is not solved **do**
    *** Collect data for the master policy ***
    $\mathcal{E} = \{\}$ ▷ Empty storage for rollouts
    for episode_id $\in \{1, \ldots,$ ppo_num_episodes$\}$ **do**
        $o_0 =$ new_episode_observation()
        $t = 0$
        while episode is not terminated **do**
            $k_t \sim \pi_m(o_t)$ ▷ Choose the skill policy
            $o_{t+n}, r_{t+n} =$ perform_skill$(\pi_s^{k_t}, o_t)$
            $t = t + n$
        $\mathcal{E} = \mathcal{E} \cup \{(o_0, k_0, r_n, o_n, k_n, r_{2n}, o_{2n}, \ldots)\}$
    *** Make a PPO step for the master policy on $\mathcal{E}$ ***
    $\mu =$ ppo_update$(\pi_m, \mathcal{E})$

of the expert actions to have zero mean and a unit variance and normalize the depth values of input frames to $[-1, 1]$. We learn BC skills using Adam [Kingma and Ba, 2014] with the learning rate 10^{-3} and a batch size 64. We also use Batch Normalization [Ioffe and Szegedy, 2015].

Task learning with RL. We learn the master policies with the PPO [Schulman et al., 2017] algorithm using the open-source implementation [Kostrikov, 2018] where we set the entropy coefficient to 0.05, the value loss coefficient to 1, and use 8 episode rollouts for the PPO update. For the RLBC method, the concatenated skill features $\{x_t^1, \ldots, x_t^K\}$ are processed with the master network $CNN(\mu)$ having 2 convolutional layers with 64 filters of size 3×3. During pretraining of skill policies we update the parameters (θ, η). When training the master policy, we only update μ while keeping (θ, η) parameters fixed. We train RLBC using 8 different random seeds in parallel and evaluate the best one.

Real robot transfer. To deploy our method on the real robot, we use the sim2real transfer technique from the previous chapter [Pashevich et al., 2019a]. This method uses a proxy task of cube position prediction and a set

of basic image transformations to learn a sim2real data augmentation function for depth images. We augment the depth frames from synthetic expert demonstrations with this method and, then, train skill policies. Once the skill policy is trained on these augmented simulation images, it is directly used on the real robot.

3.4 Experimental setup

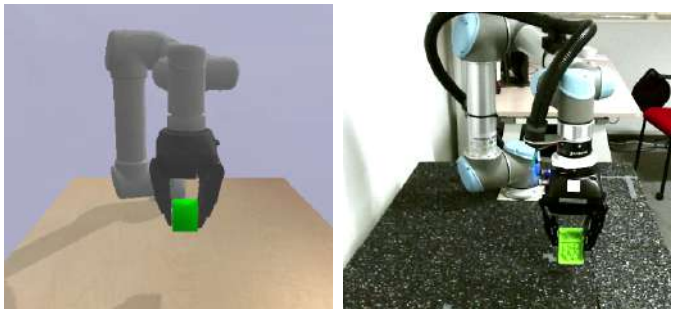
This section describes the setup used to evaluate our approach. First, we present the robot environment and the different tasks. Next, we describe the synthetic dataset generation and skill definition for each task.

Robot and agent environment For our experiments we use a 6-DoF UR5 robotic arm with a 3 finger Robotiq gripper, see Figure 3.3. In simulation, we model the robot with the `pybullet` physics simulator [Coumans, 2009]. For observation, we record depth images with the Microsoft Kinect 2 placed in front of the arm. The agent takes as input the three last depth frames $o_t \in \mathbb{R}^{224\times224\times3}$ and commands the robot with an action $a_t \in \mathbb{R}^7$. The control is performed at 10 Hz frequency.

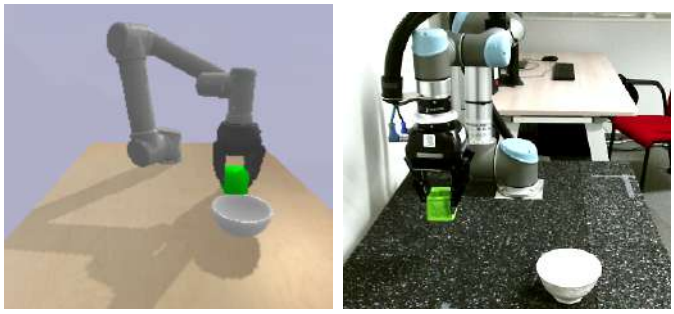
UR5 tasks For evaluation, we consider 3 tasks: UR5-Pick, UR5-Bowl and UR5-Breakfast. The UR5-Pick task picks up a cube of a size between 3.5 cm and 8.0 cm and lifts it up, see Figure 3.3a. In UR5-Bowl the robot has to grasp the cube and place it in the bowl, see Figure 3.3b. The UR5-Breakfast task contains a cup, a bottle and a bowl as shown in Figure 3.3c. We use distinct ShapeNet [Chang et al., 2015] object instances for the training and test sets (27 bottles, 17 cups, 32 bowls in each set). The robot needs to pour ingredients from the cup and the bottle in the bowl. In all tasks, the reward is positive if and only if the task goal is reached. The maximum episode lengths are 200, 600, and 2000 time-steps for UR5-Pick, UR5-Bowl, and UR5-Breakfast correspondingly.

3.4.1 Synthetic datasets

We use the simulated environments to create a synthetic training and test set. For all our experiments, we collect trajectories with random initial configurations where the objects and the end-effector are allocated within a workspace of $80 \times 40 \times 20\,\mathrm{cm}^3$. The synthetic demonstrations are collected using an expert script designed for each skill. The script has access to the full state of the system including the states of the robot and the objects.



(a) UR5-Pick



(b) UR5-Bowl

(c) UR5-Breakfast

Figure 3.3 – UR5 tasks used for evaluation: (a) task of picking up the cube, (b) task of bringing the cube to the bowl, (c) task of pouring the cup and the bottle into the bowl. (Left) simulation, (right) real robot.

To generate synthetic demonstrations, we program end-effector trajectories and use inverse kinematics (IK) to generate corresponding trajectories in the robot joints space. Each demonstration consists of multiple pairs of the three last camera observations and the robot control command performed by the expert script. For UR5-Pick, we collect 1000 synthetic demonstrated trajectories for training. For UR5-Bowl and UR5-Breakfast, we collect a training dataset of 250 synthetic demonstrations. For evaluation of each task, we use 100 different initial configurations in simulation and 20 trials on the real robot.

3.4.2 Skill definition

UR5-Pick task is defined as a single skill. For UR5-Bowl and UR5-Breakfast, we consider a set of skills defined by expert scripts. For UR5-Bowl, we define four skills: (a) go to the cube, (b) go down and grasp, (c) go up, and (d) go to the bowl and open the gripper. For UR5-Breakfast, we define four skills: (a) go to the bottle, (b) go to the cup, (c) grasp an object and pour it to the bowl, and (d) release the held object. We emphasize that the expert dataset does not contain full task demonstrations and that all our training is done in simulation. When training the RL master, we execute selected skills for 60 consecutive time-steps for the UR5-Bowl task and 220 time-steps for the UR5-Breakfast task.

3.5 Evaluation of BC skill learning

This section evaluates the different parameters of the BC skill training for the UR5-Pick task and a comparison with the state of the art. First, we evaluate the impact of the CNN architecture and data augmentation on the skill performance in Sections 3.5.1 and 3.5.2. Then, we show that the learned policies transfer to a real robot in Section 3.5.3. Finally, we compare the BC skills with the state of the art in Section 3.5.4.

3.5.1 CNN architecture for BC skill learning

Given the simulated UR5-Pick task illustrated in Figure 3.3a(left), we compare BC skill networks trained with different CNN architectures and varying number of expert demonstrations. Table 3.1 compares the success rates of policies with VGG and ResNet architectures. Policies based on the VGG architecture [Simonyan and Zisserman, 2014b] obtain success rate below 40% with 100 training demonstrations and reach 95% with 1000

Demos	VGG16-BN	ResNet-18	ResNet-101
20	1%	1%	0%
50	9%	5%	5%
100	37%	65%	86%
1000	95%	**100%**	**100%**

Table 3.1 – Evaluation of BC skills trained with different CNN architectures and number of demonstrations on the UR5-Pick task in simulation.

Demos	None	Standard	Viewpoint	Standard & Viewpoint
20	1%	49%	39%	75%
50	5%	81%	79%	93%
100	65%	97%	**100%**	**100%**

Table 3.2 – Evaluation of ResNet-18 BC skills trained with different data augmentations on UR5-Pick task in simulation.

demonstrations. ResNet [He et al., 2016] based policies have a success rate above 60% when trained on a dataset of 100 demonstrations and reach 100% with 1000 demonstrations. Overall ResNet-101 has the best performance closely followed by ResNet-18 and outperforms VGG significantly. To conclude, we find that the network architecture has a fundamental impact on the BC performance. In the following experiments we use ResNet-18 as it presents a good trade-off between performance and training time.

When examining why VGG-based BC has a lower success rate, we observe that it has higher validation errors compared to ResNet. This indicates that VGG performs worse on the level of individual steps and is hence expected to result in higher compounding errors shown in Figure 3.4.

3.5.2 Evaluation of data augmentation

We evaluate the impact of different types of data augmentations in Table 3.2. We compare training without data augmentation with 3 variants: (1) random translations, rotations and crops, as is standard for object detection, (2) record each expert synthetic demonstration from 10 varying viewpoints and (3) the combination of (1) and (2).

Success rates for UR5-Pick on datasets with 20, 50 and 100 demonstrations are reported in Table 3.2. We observe that data augmentation is particularly important when only a few demonstrations are available. For

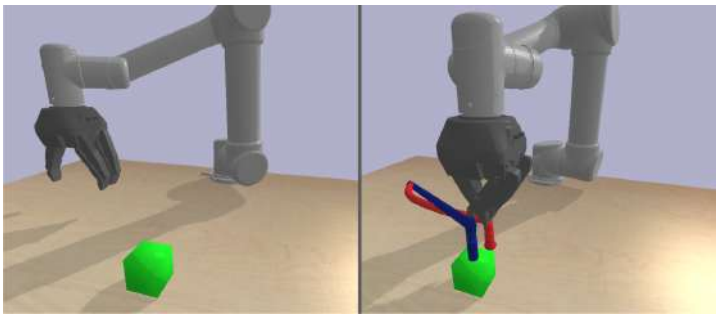


Figure 3.4 – Compounding errors from a VGG-based policy. The policy (red) drifts away from the expert trajectory (blue) and does not recover.

20 demonstrations, the policy trained with no augmentation performs at 1% while the policy trained with standard and viewpoint augmentations together performs at 75%. The policy trained with a combination of both augmentation types performs the best and achieves 93% and 100% success rate for 50 and 100 demonstrations respectively. In summary, data augmentation allows a significant reduction in the number of expert trajectories required to solve the task.

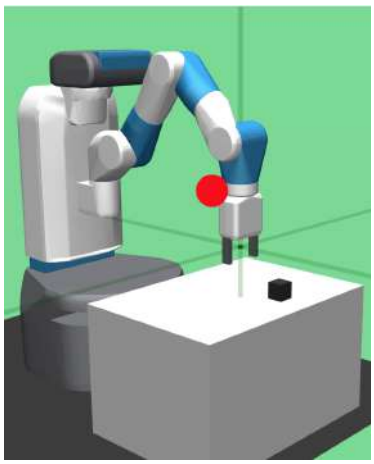
3.5.3 Real robot experiments

We evaluate our method on the real-world UR5-Pick illustrated in Figure 3.3a(right). We collect demonstrated trajectories in simulation and train the BC skills network applying standard, viewpoint and sim2real augmentations. We show that our approach transfers well to the real robot using no real images. The learned policy manages to pick up cubes of 3 different sizes correctly in 20 out of 20 trials.

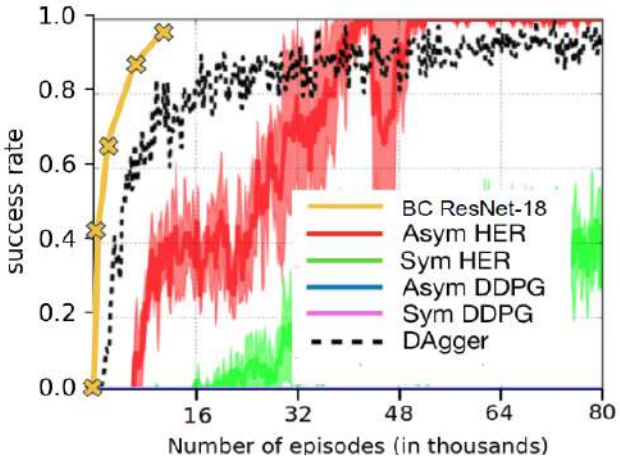
3.5.4 Comparison with state-of-the-art methods

One of the few test-beds for robotic manipulation is FetchPickPlace from OpenAI Gym [Plappert et al., 2018] implemented in `mujoco` [Todorov et al., 2012], see Figure 3.5a. The goal for the agent is to pick up the cube and to move it to the red target (see Figure 3.5a). The agent observes the three last RGB-D images from a camera placed in front of the robot $o_t \in \mathbb{R}^{100\times100\times4\times3}$. The positions of the cube and the target are set at random for each trial. The reward of the task is a single sparse reward of success. The maximum length of the task is 50 time-steps.

For a fair comparison with [Pinto et al., 2018], we do not use any data augmentation. We report the success rate of ResNet-18 policy in Fig-



(a) FetchPickPlace



(b) Comparison with [Pinto et al., 2018]

Figure 3.5 – Comparison of BC ResNet-18 with state of the art [Pinto et al., 2018] on the FetchPickPlace task. BC ResNet-18 results are reported for 200 different initial configurations.

ure 3.5b. We follow [Pinto et al., 2018] and plot the success rate of both RL and IL methods with respect to the number of episodes used (either trial episodes or demonstrations). Our approach outperforms the policies trained with an imitation learning method DAgger [Ross and Bagnell, 2014] in terms of performance and RL methods such as HER [Andrychowicz et al., 2017] and DDPG [Lillicrap et al., 2016] in terms of data-efficiency. According to [Pinto et al., 2018], DAgger does not reach 100% even after $8 * 10^4$ demonstrations despite the fact that it requires an expert during training. HER reaches the success rate of 100% but requires about $4 * 10^4$ trial episodes. Our approach achieves the 96% success rate using 10^4 demonstrations.

Our policies differ from [Pinto et al., 2018] mainly in the CNN architecture. Pinto et al. [Pinto et al., 2018] use a simple CNN with 4 convolutional layers while we use ResNet-18. Results of this section confirm the large impact of the CNN architecture on the performance of visual BC policies, as was already observed in Table 3.1.

3.6 Evaluation of RLBC

This section evaluates the proposed RLBC approach and compares it to baselines introduced in Section 3.6.1. First, we evaluate our method on UR5-Bowl in Section 3.6.2. We then test the robustness of our approach to

UR5-Bowl perturbations	Detect & Plan	Detect & Replan	BC-ordered	RLBC
No perturbations	17/20	16/20	17/20	**20/20**
Moving objects	0/20	12/20	13/20	**20/20**
Occlusions	17/20	10/20	2/20	**18/20**
New objects	16/20	14/20	15/20	**18/20**

Table 3.3 – Comparison of RLBC with 3 baselines on the real-world UR5-Bowl task with dynamic changes of the cube position, dynamic occlusions and new object instances.

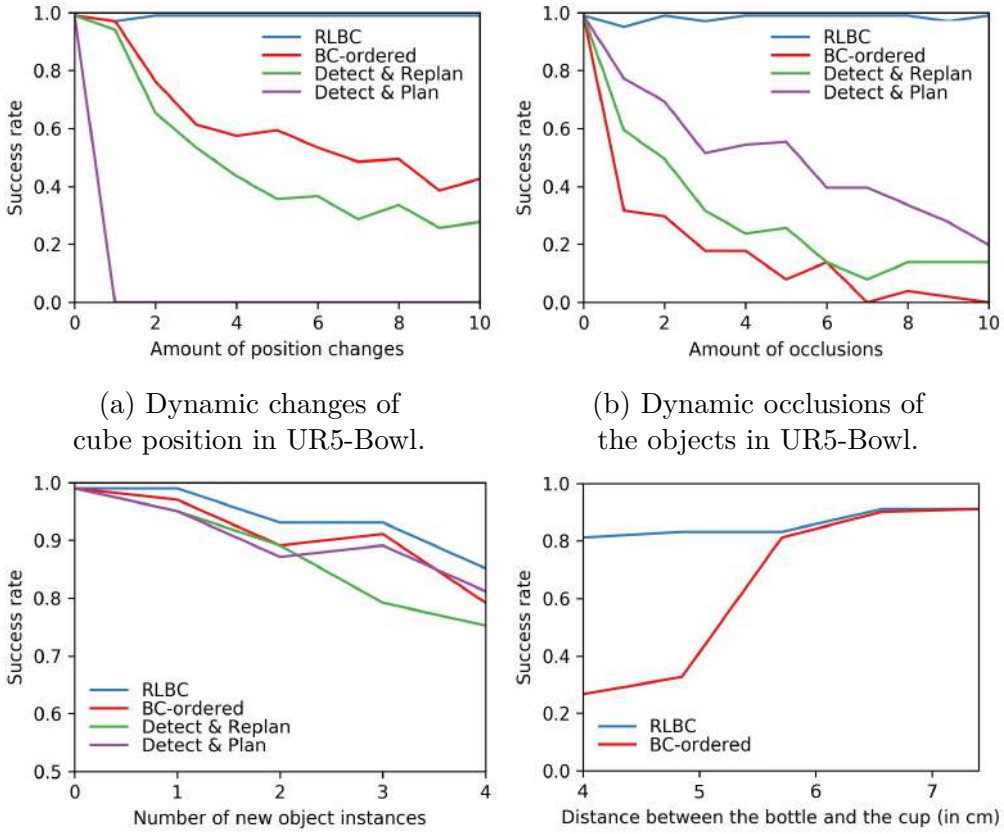
various perturbations such as dynamic changes of object positions, dynamic occlusions, unseen object instances and the increased probability of collisions due to small distances between objects. In Section 3.6.3, we show that RLBC outperforms the baselines on those scenarios both in simulation and on a real robot. Note, that our real robot experiments are performed with skills and master policies that have been trained exclusively in simulation using sim2real augmentation [Pashevich et al., 2019a]. We use the same policies for all perturbation scenarios. Qualitative results of our method are available at the project website [Strudel et al., 2020a].

3.6.1 Baseline methods

We compare RLBC with 3 baselines: (a) a fixed sequence of BC skills following the manually pre-defined correct order (BC-ordered); (b) an open-loop controller estimating positions of objects and executing an expert script (Detect & Plan); (c) a closed-loop controller performing the same estimation-based control and replanning in case if object positions change substantially (Detect & Replan). We use the same set of skills for RLBC and BC-ordered. We train the position estimation network using a dataset of 20.000 synthetic depth images with randomized object positions. All networks use ResNet-18 architecture and are trained with the standard, viewpoint and sim2real augmentations described in Section 3.5.2.

3.6.2 Results on UR5-Bowl with no perturbations

We first evaluate RLBC and the three baselines on the UR5-Bowl task (see Figure 3.3b). When tested in simulation, all the baselines and RLBC manage to perfectly solve the task. On the real-world UR5-Bowl task, BC-ordered and Detect & Plan baselines sometimes fail to grasp the object



(a) Dynamic changes of cube position in UR5-Bowl.

(b) Dynamic occlusions of the objects in UR5-Bowl.

(c) Replacing the cube with unseen objects in UR5-Bowl.

(d) Decreasing the distance between objects in UR5-Breakfast.

Figure 3.6 – Performance of RLBC and baseline methods in simulated environments under perturbations: (a) dynamic changes of cube position; (b) dynamic occlusions; (c) replacing the cube by unseen objects; (d) decreasing the distance between objects.

which leads to task failures (see Table 3.3, first row). On the contrary, RLBC solves the task in all 20 episodes given its ability to re-plan the task in the cases of failed skills.

We have also attempted to solve the simulated UR5-Bowl task without skills by learning an RL policy performing low-level control. We have used ImageNet pretrained ResNet-18 to generate visual features. The features were then used to train low-level RL control policy with PPO. Whereas such a low-level RL policy did not solve the task a single time after 10^4 episodes, RLBC reaches 100% after 400 episodes.

3.6.3 Robustness to perturbations

Robustness to dynamic changes in object position. We evaluate RLBC against the baselines in the UR5-Bowl scenario where the cube is moved several times during the episode. We plot success rates evaluated in simulation with respect to the number of position changes in Figure 3.6a. We observe the stability of RLBC and the fast degradation of all baselines. As both RLBC and BC-ordered use the same set of skills, the stability of RLBC comes from the learned skill combination. The "Moving objects" row in Table 3.3 reports results for 3 moves of the cube evaluated on the real robot. Similarly to the simulated results, we observe excellent results of RLBC and the degraded performance for all the baselines.

Robustness to occlusions. We evaluate the success of UR5-Bowl task under occlusions. Each occlusion lasts 3 seconds and covers a large random part of the workspace by a cardboard. Figure 3.6b shows success rates with respect to the number of occlusions in the simulated UR5-Bowl environment. Similarly to the perturbation results in Figure 3.6a, RLBC demonstrates high robustness to occlusions while the performance of other methods quickly degrades. The "Occlusions" row in Table 3.3 reports results for a single occlusion performed during the real-robot evaluation. Baseline methods are strongly influenced by occlusions except Detect & Plan which performs well unless occlusion happens during the first frames. Our RLBC policy performs best compared to other methods.

Robustness to new object instances. We evaluate the robustness of methods to the substitution of a cube by other objects not seen during the training of UR5-Bowl task. Figure 3.6c shows the success rate of RLBC and other methods with respect to the number of new objects in simulation. The novel objects are ordered by their dissimilarity with the cube. The difficulty

Figure 3.7 – Illustration of a failure case for the BC-ordered approach for UR5-Breakfast with close by objects.

UR5-Breakfast cup distance	BC-ordered	RLBC
> 10 cm	**16/20**	**16/20**
< 4 cm	8/20	**16/20**

Table 3.4 – Comparison of RLBC and manually ordered BC skills on simple and hard scenarios of the UR5-Breakfast task.

of grasping unseen objects degrades the performance of grasping skills. In contrast to other methods RLBC is able to automatically recover from errors by making several grasping attempts. Table 3.3 reports corresponding results on a real robot where the cube has been replaced by 10 unseen objects. Similarly to the other perturbations we observe superior performance of RLBC.

Impact of the distance between objects. We vary the distance between a bottle and a cup in the UR5-Breakfast task. The smaller distance between objects A and B implies higher probability of collision between a

robot and A when grasping B behind A. The choice of the grasping order becomes important in such situations. While our method is able to learn the appropriate grasping order to maximize the chance of completing the task, the BC-ordered and other baselines use pre-defined order. Figure 3.6d demonstrates the performance of RLBC and BC-ordered for different object distances in the simulated UR5-Breakfast task. As expected, RLBC learns the correct grasping order and avoids most of collisions. The performance of BC-ordered strongly degrades with the decreasing distance. In the real-world evaluation shown in Table 3.4, both RLBC and ordered skills succeed in 16 out of 20 episodes when the distance between objects is larger than 10 cm. However, the performance of BC-ordered drops to 8/20 when the cup and the bottle are at 4cm from each other. In contrast, RLBC chooses the appropriate object to avoid collisions and succeeds in 16 out of 20 trials. Figure 3.7 shows an example scene with a cup and a bottle being near to each other while the cup is placed in front of the bottle. As the BC-ordered policy is pre-programmed to grasp the cup first, the execution of this policy results in a collision between a gripper and a bottle, followed by the failure of the task. Our RLBC policy learns to select the order of objects for grasping to avoid failures of the task.

3.7 Qualitative results

We present additional qualitative results for the RLBC approach on the real robot. We first illustrate examples of UR5-Bowl and UR5-Breakfast policies while the robot is facing the challenges of previously unseen objects, dynamic changes of object locations and occlusions. We then illustrate feature map activations of the network providing better understanding of learned policies.

UR5-Bowl: multiple objects. We experiment with the RLBC policy trained in the UR5-Bowl environment. Once the robot succeeds to place a cube in the bowl, we put another cube on the table and let the policy continue, see Figure 3.8. While the UR5-Bowl policy has been trained to handle one cube only, it automatically generalizes to multiple cubes when run in a loop.

UR5-Bowl: previously unseen objects. We further test the RLBC UR5-Bowl policy in the presence of previously unseen objects. While the policy has been trained to manipulate cubes of different sizes, we observe its robustness to other object shapes. As shown in Figure 3.9, the policy

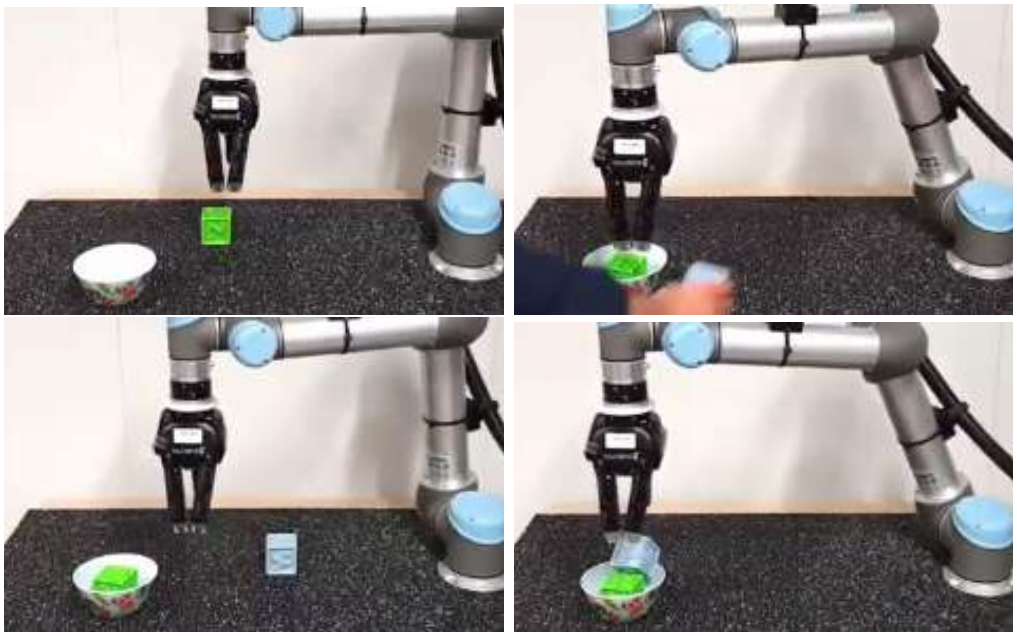


Figure 3.8 – RLBC approach for UR5-Bowl with two cubes.

successfully grasps and places into a bowl real objects, such as apples, oranges, lemons, and toys. Notably, in cases of failing to grasps an object, the robot automatically recovers and completes the task. This behavior comes naturally from our RLBC master policy that has learned to adapt the sequence of skills given current observations of the scene.

UR5-Breakfast: new object instances. To enable generalization of learned policies to new object instances, our UR5-Breakfast environment contains cups, bottles and bowls of different shapes from ShapeNet [Chang et al., 2015]. During testing we run the learned RLBC UR5-Breakfast policy on a real robot and experiment with instances of bottles and cups unseen during training. Figure 3.10 demonstrates successful executions of the RLBC UR5-Breakfast policy in scenes with significant variations in object shapes, for example, using a wine glass instead of a cup.

UR5-Breakfast: dynamic changes of object location. Our BC skills make decisions at every time-step and, hence, can instantly adapt to changing conditions of the scene. We verify this by varying object positions during grasping attempts of the RLBC UR5-Breakfast policy. Figure 3.11 illustrates the reactive behavior of the robot grasping a cup that is being

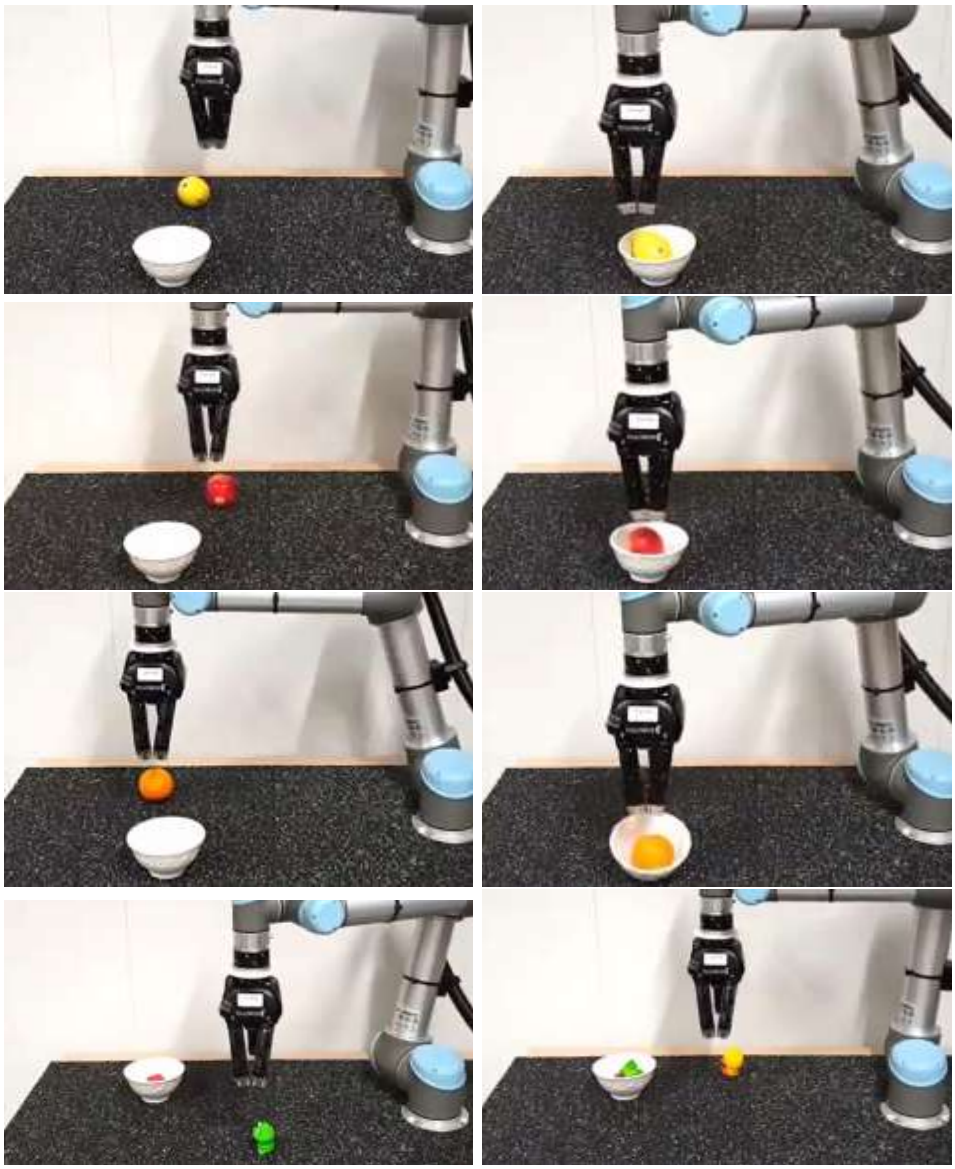


Figure 3.9 – RLBC approach for UR5-Bowl with previously unseen objects.

simultaneously moved by the person. The cup is successfully grasped after multiple changes of its position.

UR5-Breakfast: occlusion. Another example of the instant re-planning by our RLBC policy is demonstrated in Figure 3.12. While the robot approaches an object, we temporary occlude the object and disrupt the executing BC skill. Given the hierarchical nature of our RLBC approach,

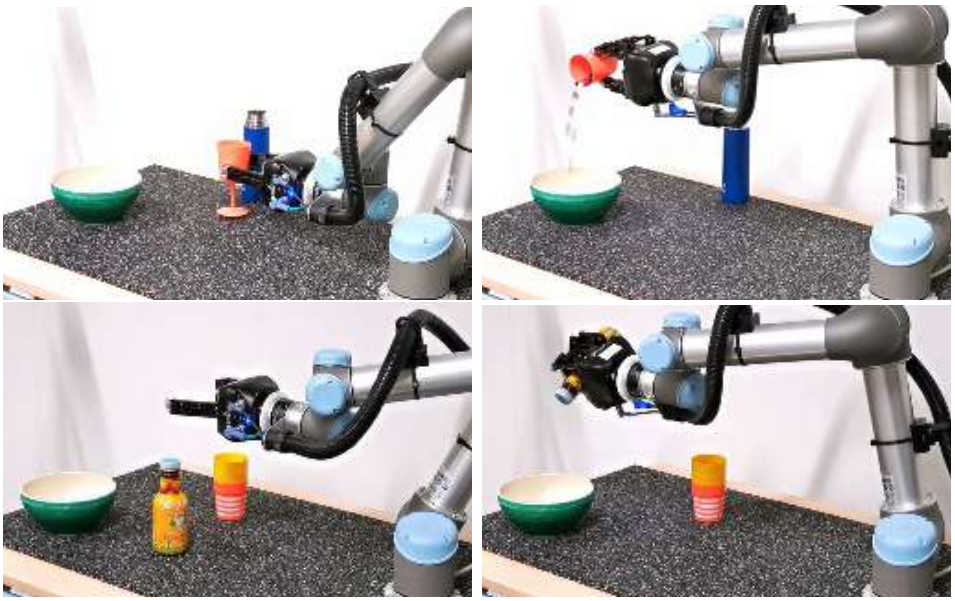


Figure 3.10 – RLBC approach for UR5-Breakfast with previously unseen object instances.

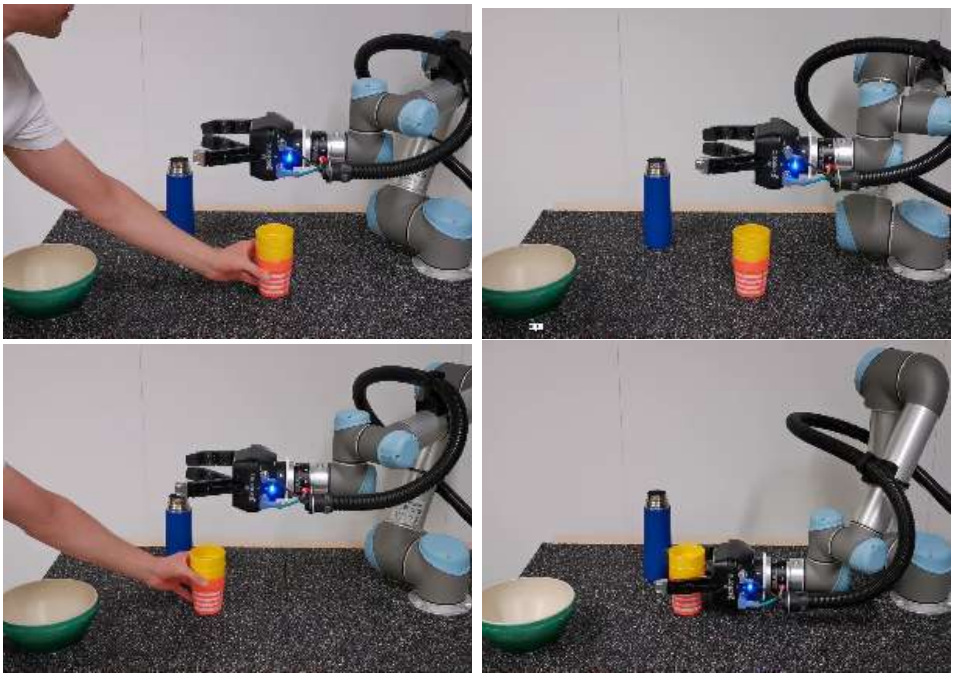


Figure 3.11 – RLBC approach for UR5-Breakfast with dynamic changes of object locations.

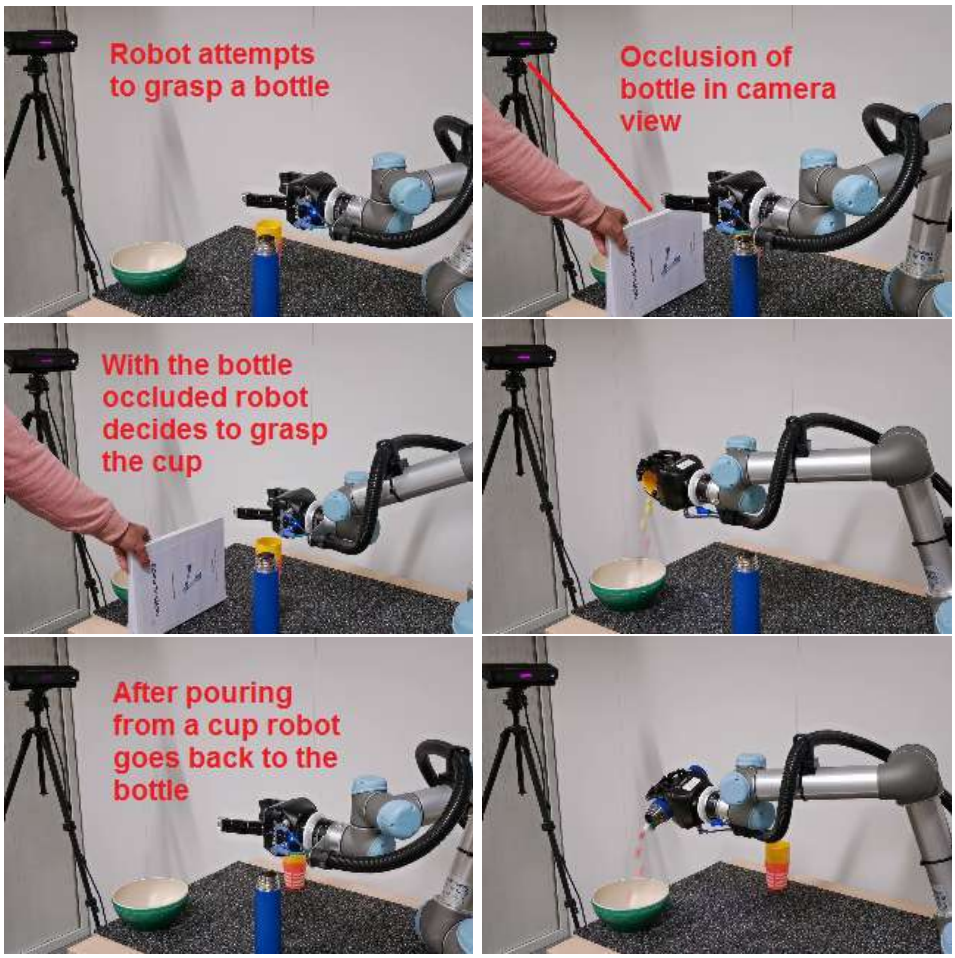


Figure 3.12 – RLBC UR5-Breakfast policy executed in a robot scene with a temporary occluded bottle.

the RL master is able to recover from this failure by starting another skill that leads to the completion of the task. More precisely, when the bottle gets occluded in the example of Figure 3.12, the robot changes its strategy and decides to grasp and pour from a cup. Once the occlusion is removed, the robot automatically resumes and completes the task by grasping and pouring from the bottle.

Feature map activations The RLBC policy uses no explicit representation of scenes, for example in terms of categories and locations of objects. Some interpretation of learned policies, however, can be obtained by examining spatial activations of the neural network at intermediate network layers. Figure 3.13 shows silency maps of an RLBC policy highlighting

grasping a bottle

moving a bottle

pouring from a bottle

releasing a bottle

Figure 3.13 – Left: Feature map activations of the RLBC UR5-Breakfast policy are overlayed on the input depth images. Right: corresponding frames taken from a different viewpoint.

which parts of the image the agents concentrates on. The silency maps are computed as activations of convolutional feature maps obtained from last layers of $CNN(\theta)$ and $CNN(\eta^i), i = 1, \ldots, K$ (see Figure 3.2), averaged over all channels and layers. The resulting heatmaps are shown for different stages of the UR5-Breakfast task. Interestingly, while grasping and moving the bottle, the network generates highest activations around the bottle and the gripper, while ignoring other objects. When releasing the bottle, however, high activations are also observed around the cup. The attention to the cup might be explained by the need of avoiding collisions when placing the bottle on the table. Note, that we provide no intermediate rewards to RL, however, RL learns to avoid collisions since collisions imply failures of the final task and, hence, no final positive reward during training. Once the bottle is placed on the table, activations become low for the bottle, while the heatmap obtains maximum values for the manipulated cup. Observations of such feature maps have been useful in our work to identify certain cases of failures. We believe feature map activations are a useful tool to interpret learned policies.

3.8 Conclusion

This chapter presents a method to learn combinations of primitive skills. We propose to pretrain skills using behavior cloning and demonstrate how to use recent CNN architectures and data augmentation to significantly decrease the number of required demonstrations. We then propose to learn a master policy that switches between skills using reinforcement learning and a task completion signal. In contrast to previous methods, our approach requires neither full-task demonstrations nor intermediate rewards. We use our sim2real method proposed in Chapter 2 and transfer policies learned in simulation to a real robot using only synthetic data. We demonstrate excellent results both in simulation and on a real robot. Due to the ability of skills to react on dynamic changes in the scene and the ability of the master policy to replan in case of a skill failure, we demonstrate the versatility of our approach in challenging real-world settings which include changes of object positions, temporary object occlusions, and new object instances.

4

Learning visual policies for building 3D shape categories

4.1 Introduction

Our daily physical activities such as cooking, dressing or navigation require complex sequences of actions which people successfully and seamlessly plan based on sensory input. Action planning typically depends on the goal and constraints provided by the environment. Despite extensive prior work, existing autonomous agents are still far from the human-level planning performance, especially in unknown and cluttered environments [Ebert et al., 2018, Wang et al., 2019a].

Action planning is a hard problem due to the large action spaces, exponential complexity and partial observability [Irpan, 2018, Jaderberg et al., 2017]. To simplify the problem, existing work on task planing [Garrett et al., 2018b, Nair et al., 2018, Srivastava et al., 2014, Toussaint, 2015] typically operates in the state space assuming the full knowledge of the environment. While such an assumption can be practical in structured and controlled environments, full state reconstruction for common scenes remains a highly challenging problem [Grabner et al., 2018]. Arguably, the precise recovery of scene parameters such as its geometry, composition, friction coefficients, etc., is more difficult than the primary planning task itself. It is therefore desirable to design sensor-based planning policies that do not rely on explicit scene geometry and full state estimation.

Vision-based control policies have recently become popular for robotic manipulation [Agrawal et al., 2016, Levine et al., 2015, Zeng et al., 2019] and navigation [Codevilla et al., 2018, Gandhi et al., 2017]. While this line of work shows promise, it has mostly been applied to the low-level motion planning such as predicting next motion direction. In our work we aim

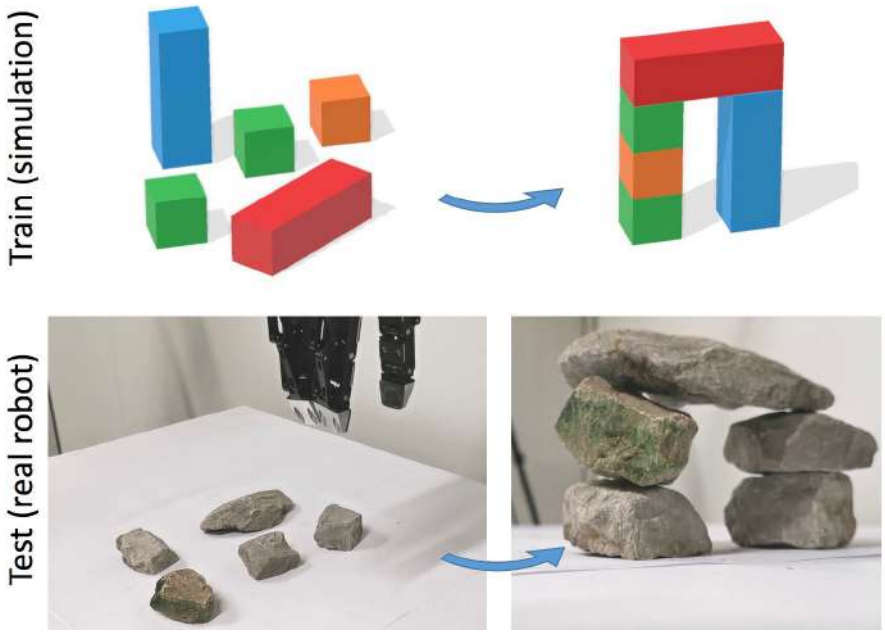
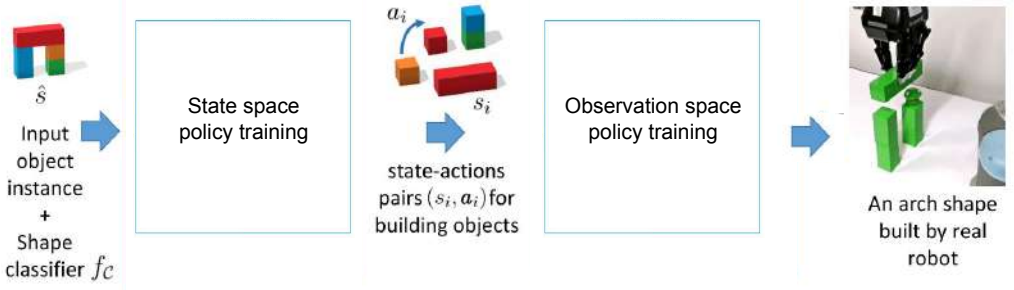


Figure 4.1 – Primitives on the left are assembled by our learned policy into arches on the right. We assemble objects of similar shapes in the simulator and learn visual manipulation policies that can build real 3D shapes from unseen primitives resembling building blocks used during training.

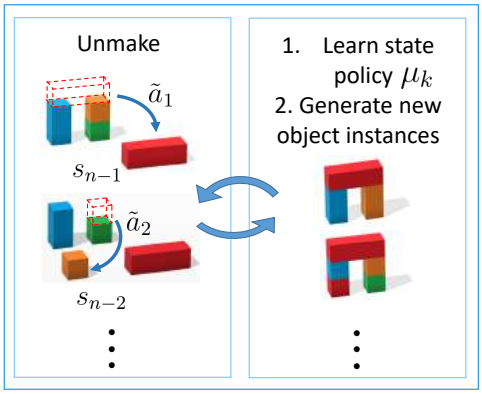
to learn visual policies for high-level task planning. Given visual input, our policies generate sequences of picking, rotating and placing actions for building 3D shapes.

Object manipulation has a long history in robotics. In particular, assembling objects from a given set of primitives has been addressed, for example, in [Popov et al., 2017, Chen et al., 2019b]. Prior work in this domain often aims to build particular object instances for which the structure is pre-defined [Suárez-Ruiz et al., 2018], specified by demonstrations [Huang et al., 2018, Duan et al., 2017] or given by a goal image [Janner et al., 2019]. Here we go further and learn to assemble different objects of a shape category. Such a task is significantly more complex compared to building particular object instances as it requires generalization to varying sets of building primitives. Moreover, we show empirically that our method is able to generalize to new primitives unseen during training (see Figure 4.1).

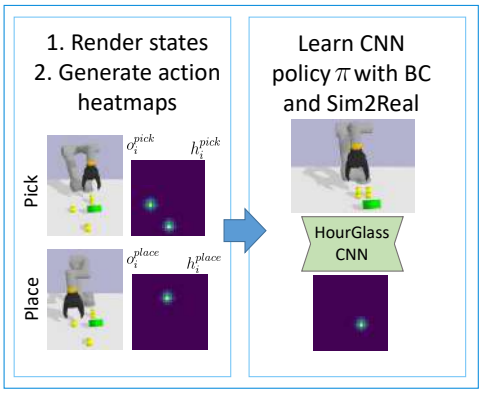
Our approach contains two stages as illustrated in Figure 4.2a. In the first stage we discover new object instances and learn their assembling policies in state space. To this end, we propose a disassembling procedure and



(a) Our two-phase method overview. Given an example object and a shape classifier on the left, our method generates new objects with similar shape and discovers action sequences for building these objects in the state space. The state and observation space policy training phases are explained below.



(b) State space policy training



(c) Observation space policy training

Figure 4.2 – The first phase of the method works in the low-dimensional state space and generates state-action pairs $(s_i, \boldsymbol{a}_i)$ using the proposed disassembly procedure iteratively (see Figure 4.2b). The second phase of the method works in the visual observation space and trains a control policy for a real robot (see Figure 4.2c). It first renders states s_i as realistic observations o_i and generates 2D heatmaps $(h_i^{\text{pick}}, h_i^{\text{place}})$ encoding source and target locations and orientations of one or several primitives. Heatmaps can represent multiple hypothesis for the next action when several identical primitives are used or multiple object instances can be assembled. Positions on our 2D heatmaps correspond to positions on the 2D surface of a table, hence, the identified local maxima on heatmaps can be used to control the robot. It then trains a Behavior Cloning policy π to predict h^{pick} and h^{place} from observations. The learned policy is directly transferred to a real robot which assembles 3D objects from primitives.

generate assembly trajectories by (a) unbuilding objects and (b) reverting action sequences. We then learn a value function and apply it to build new object instances. We iterate the disassembling and learning steps to obtain a policy assembling a 3D shape.

In the second stage we assemble objects given images of observed scenes. We render states from assembly trajectories obtained in the first stage and learn visual assembly policies with Behavior Cloning (BC) [Pomerleau, 1989]. To enable predictions of multiple valid actions at any given time, we propose a heatmap representation for the output of visual policies. While all our policies are learned in a simulated environment, we enable their direct transfer to a real robot using the sim2real augmentation technique presented in Chapter 2 [Pashevich et al., 2019a].

As main contributions of this chapter, we (i) propose a novel disassembly algorithm for building shape categories in state space, (ii) design and learn visual policies with heatmap outputs to address multimodality of predicted actions, (iii) demonstrate a successful application of the method to a new task of building shape categories on a real UR5 robot. Moreover, our policies are learned with no human demonstrations, can re-assemble partially built objects, and adapt to unseen primitives resembling building blocks used during training.

4.2 Related work

Assembly tasks such as constructing IKEA furniture [Suárez-Ruiz et al., 2018] remain to be a hard robotics challenge. Learning-based methods usually address simpler tasks such as cube stacking [Riedmiller et al., 2018, Pashevich et al., 2018]. Duan et al. [Duan et al., 2017] use demonstrations and attention modules to build a tower instance shown by an expert. Janner et al. [Janner et al., 2019] learn an object-centric representation of the scene to reproduce a tower instance from a goal image with an MPC-like control. Huang et al. [Huang et al., 2018] train a Graph Network to build a tower instance specified by a demonstration. We go beyond specific object instances and aim to assemble multiple objects from a given shape category, such as an arch. Moreover, we learn to build objects from different sets of possibly unseen primitives. To facilitate the learning, we propose to use disassembling to generate assembly trajectories. While the idea of reversible actions has been explored e.g., in [Florensa et al., 2017b, Nair et al., 2020, Sukhbaatar et al., 2017, Hosu and Rebedea, 2016], our method differs from the work on disassembling object instances [Zakka et al., 2019] by computing multiple disassembly paths and accounting for alternative

valid actions.

Our work is related to methods of Task and Motion Planing (TAMP) [Garrett et al., 2015, Lozano-Pérez and Kaelbling, 2014, Srivastava et al., 2014, Toussaint, 2015]. Long-term task planning prohibits costly rollouts, hence, TAMP methods deploy preconditions and postconditions to actions and optimize symbolic planners [Fikes and Nilsson, 1971]. While some of these methods solve impressive tasks, conditions require manual and task-specific design [Edelkamp and Hoffmann, 2004, He et al., 2015, Paxton et al., 2019]. Moreover, TAMP methods typically operate in state-space [Garrett et al., 2018a], hence, their generalization to sensor-based input in the real world requires non-trivial scene understanding [Dantam et al., 2018]. In our work we learn visual policies and directly predict control sequences from image inputs.

Convolutions Neural Networks (CNNs) have significantly advanced visual recognition [He et al., 2016, Ren et al., 2015, Newell et al., 2016] and robotics, for example in tasks such as tossing objects [Zeng et al., 2019], cube stacking [Popov et al., 2017], grasping [Lampe and Riedmiller, 2013] and opening doors [Gu et al., 2016]. Direct methods for visual control avoid explicit scene reconstruction and derive actions directly from image observations. Such methods typically use Reinforcement Learning (RL) [Kober et al., 2013] with auxiliary rewards [Riedmiller et al., 2018] or Imitation Learning (IL) [Argall et al., 2009, Sadeghi and Levine, 2017] relying on large amounts of demonstrations [Zhang et al., 2018, Rahmatizadeh et al., 2018]. The complexity of problems addressed by direct methods is typically limited by the task length and the number of manipulated objects [Riedmiller et al., 2018, Zhang et al., 2018, Rahmatizadeh et al., 2018]. Indirect methods first estimate scene parameters [Espiau et al., 1992, Andrychowicz et al., 2017] such as object positions and orientations, and then deploy state-based planning strategies. Scene reconstruction from images, however, might be a more challenging task than solving a control task itself [Pinto and Gupta, 2016, Grabner et al., 2018]. We avoid drawbacks of direct and indirect methods and first solve the task in the simulated state space. We then use obtained solutions as automatic supervision for learning visual policies in the observation space. Inspired by [Levine et al., 2015, Mahler et al., 2017], we render states and train visual policies for a real robot using BC [Pomerleau, 1989] and sim2real [Pashevich et al., 2019a, Tobin et al., 2017].

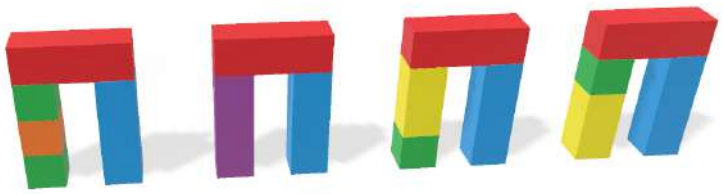


Figure 4.3 – Examples of different shape instances that belong to an arch shape category of 4 units height. While the first three instances are built of different primitives, the third and the fourth instances are built of the same set of primitives but differ in their configurations.

4.3 Approach

We address the problem of building a 3D object shape by manipulating a set of available primitives with a robot. The configuration of the primitives on the table defines the state, $s \in \mathcal{S}$. We assume to have access to a shape classifier function $f_{\mathcal{C}} : \mathcal{S} \rightarrow \{0, 1\}$. The classifier decides whether a given state belongs to the target shape category which is defined as a subset of the state space, $\mathcal{C} = \{s \in \mathcal{S} | f_{\mathcal{C}}(s) = 1\}$. The shape category includes several shape instances which differ either in which primitives are used or in the assembled configuration of primitives as shown in Figure 4.3. Given $f_{\mathcal{C}}$ and a single shape instance $\hat{s} \in \mathcal{C}$, our method learns a visual policy π that generates a robot action given a camera observation $o \in \mathcal{O}$. The resulting sequence of actions is then used to assemble a shape instance from available primitives. Our policy operates in the observation space $\mathcal{O}$, i.e., it only has access to the image of a current scene before deciding on the next action. Moreover, the policy is expected to build new object configurations from unseen primitives that resemble building blocks used during training.

4.3.1 Overview of the method

Our method has two stages: (i) generating action sequences in the state space and (ii) learning visual policies in the observation space. We use a simulated environment for both stages, however, our visual policies are trained with sim2real data augmentation presented in Chapter 2 [Pashevich et al., 2019a] and directly transfer to the real robot. The overview of the method is presented in Figure 4.2.

The first stage aims to find new shape instances and to construct action sequences for building them. It takes as input one 3D shape and a shape classifier. We propose to use an *unmake* procedure to generate valid ac-

tion sequences. We disassemble the given shape instance and interpret the resulting sequences as reversed assembly demonstrations. We disassemble objects in multiple ways to find all assembly actions that are possible in the same state. We refer to the shape classifier $f_{\mathcal{C}}$ as a sparse reward signal which we use to learn a state-value function V_k. We generate new shape instances using a state policy μ_k which is greedy with respect to the learned V_k. For a fixed number of iterations, we repeat the unmake procedure using the new set of instances and train an updated value function V_{k+1}. This part of our approach is described in Section 4.3.2.

In the second stage we learn a visual policy that infers appropriate actions from image observations. We convert states s_i into observations $o_i = R(s_i)$ using a graphics renderer R and train a CNN policy π with Behavior Cloning (BC). Given the assembly trajectories produced by the first stage, each state is associated with a valid set of actions $\boldsymbol{a}_i$ that we turn into a heatmap h_i to predict all possible actions simultaneously. Then π is trained in a supervised manner using observation-heatmap pairs (o_i, h_i). This part of our approach is described in Section 4.3.3.

4.3.2 Building objects in state space

We define the full state of our environment by the vector $s = (x^1, \ldots, x^m)^\top$ with $x \in \mathbb{R}^{12}$ representing parameters for m primitive shapes (our building blocks) in the scene. Each primitive x is defined by three position coordinates, three orientation angles in the 3D space, three spatial extents (width, height and depth), and three color channels (the building order may depend on the color, see Section 4.4.2). A robot action $a \in \mathcal{A}$ corresponds to a high-level skill of picking, placing and rotation of a primitive: $a = (x, p, o)$ where $x \in \{x^1, \ldots, x^m\}$ is the primitive to pick, $p \in \mathbb{R}^3$ is the position to place it, and $o \in \mathbb{R}^3$ is the orientation x is placed in. We restrict orientations of primitives to the three axis-parallel directions and assume that all primitives are located on the surface of a table or on top of each other. Assuming access to the simulator T, applying action a_t in the state s_t would result into $s_{t+1} = T(s_t, a_t)$.

Building an object from a given set of primitives requires finding an appropriate sequence of actions $a_{1:n} = \{a_1, \ldots, a_n\}$ that transforms an initial state s_0 into the desired shape state $s_n \in \mathcal{C}$. Finding a correct sequence $a_{1:n}$ is not trivial even in the state space. Given the large space of possible actions and the exponential growth of the number of action sequences depending on n, the naive brute-force search works only for building simple objects.

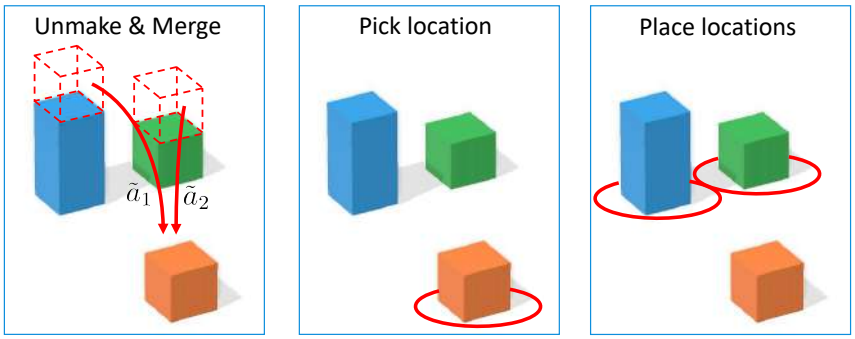


Figure 4.4 – The proposed unmake procedure disassembles objects in multiple ways (left) and generates set of pick & place actions available from each state (right). For visual policies, we use the heatmap representation to predict all the actions simultaneously.

Making objects by unmaking. For an object example defined by the state $\hat{s} \in \mathcal{C}$ we propose to find valid building sequences $a_{1:n}$ via disassembling or *unmaking* $\hat{s}$. We first find a sequence of unmake actions $\tilde{a}_{1:n} = \{\tilde{a}_1, \ldots, \tilde{a}_n\}$, where $\tilde{a}_i$ moves a random non-blocked primitive on an object * to a random non-occupied location on the table. If m is the number of primitives constituting object $\hat{s}$, we can disassemble $\hat{s}$ by a sequence of $n = m - 1$ unmake actions. We call an action invertible if for any action $\tilde{a}$ such that $T(s_i, \tilde{a}) = s_j$ there exists an inverse action $a = \tilde{a}^{-1}$ such that $T(s_j, a) = s_i$. We assume our unmake actions to be invertable and obtain a valid sequence of actions for building $\hat{s}$ as $a_{1:n} = \{\tilde{a}_n^{-1}, \ldots, \tilde{a}_1^{-1}\}$.

We use the above randomized unmake procedure to generate a large and diverse set of valid state-action sequences. While each action a_i in the sequences is associated with a single state s_i, there might be *multiple valid assembly actions* in the state s_i which would lead to a correct shape instance in the future. For example, when building an arch, the same cube could be placed both to the left and to the right pillars (see Figure 4.4). We do an additional step to associate each state s_i with a set of valid actions available in it. We look for identical states in the generated state-action sequences and merge their actions into a set of valid actions. However, the states may not match exactly due to the randomness in the initial positions of primitives. Therefore, we merge actions of *equivalent* states (denoted as $\approx$) which we define as states that differ only in positions of primitives located

∗. By "non-blocked primitives on objects" we refer to primitives that are not on the table surface and that can be freely lifted above their current positions. Such primitives can be easily derived from s.

on the table. As a result, we record state-actions pairs $(s_i, \boldsymbol{a}_i)$ where actions $\boldsymbol{a}_i = \{a_j | s_j \approx s_i\}$ are used for generating heatmaps in the observation space phase.

Given M action sequences of length N we collect a dataset $\mathcal{D}_\mu$ with state-actions pairs $\mathcal{D}_\mu = \{(s_i, \boldsymbol{a}_i)\}_{i=1,\ldots,M*N}$ that can be readily used to train a policy for assembling an object instance defined by state $\hat{s}$.

Finding new object instances. To generalize our policies to build new objects of the similar shape given any set of primitives, we construct new instances through learning a value function. The value function $V_k : \mathcal{S} \to \mathbb{R}$ estimates the sum of discounted rewards $r : \mathcal{S} \to \mathbb{R}$ under an assembly policy $\mu_k : \mathcal{S} \to \mathcal{A}$ where $V_k(s) = \mathbb{E}_{\mu_k}\left[\sum_j \gamma^j r(s_{t+j}) | s_t = s\right]$ and $\gamma \in [0, 1)$ is the discount factor. We define the reward with the shape classifier $r(s_t) = f_{\mathcal{C}}(s_t)$ which makes the value function learn the actions required to build the shape. Given a learned value function V_k, we deduce the assembly policy by choosing a_{t+1} as $\mu_{k+1}(s_t) = \arg\max_a V_k(T(s_t, a))$. We sample an initial state s_0 with a random set of primitives and apply μ_{k+1} to choose an action sequence $a_{1:n}$ resulting in an object of the desired shape: $f_{\mathcal{C}}(s_n) = 1$. We record all instances of the target shape found by assembling is a set $\mathcal{C}_k^V$.

Learning value function. The value function V_k is learned iteratively using instances found with the greedy policy μ_{k-1}. In the first stage of training, we learn V_0 using the input instance $\hat{s}$ only. We run the unmake procedure for all discovered instances in $\mathcal{C}_k^V$, Given a sequence of state-actions pairs $\{(s_1, \boldsymbol{a}_1), \ldots, (s_n, \boldsymbol{a}_n)\}$, the value function estimate for state s_i is $\hat{V}(s_i) = \gamma^{n-i}$. We also estimate values of states obtained by applying random disassembly actions $\tilde{a}_j$ to trajectory states s_i: $s_i^j = T(s_i, \tilde{a}_j)$. The value estimate $\hat{V}(s_i^j)$ is known if there exists $s_j \in \{s_1, \ldots, s_n\}$ such that $s_i^j \approx s_j$. Otherwise, we set the value as $\hat{V}(s_i^j) = \gamma \hat{V}(s_i)$ which means that we can reach s_i from s_i^j using a single assembly action $\tilde{a}_j^{-1}$. We record all state-value pairs to the dataset $\mathcal{D}_V^k$ and learn the value function by minimizing the loss

$$\hat{\eta}_k = \arg\min_\eta \mathrm{MSE}(V_k(s_i), \hat{V}(s_i)), \tag{4.1}$$

where V_k is implemented as a fully connected neural network with parameters $\hat{\eta}_k$ and MSE is the mean square error. Once the training is converged, we discover new shape instances with μ_k, unmake them, recollect $\mathcal{D}_V^{k+1}$ and learn V_{k+1}. After K phases, we run the unmake procedure on the set of discovered instances $\mathcal{C}_K^V$ and record all state-actions pairs to the dataset $\mathcal{D}_\mu$.

The overview of our approach in the state space is illustrated by Figure 4.2b and lines 14-28 of Algorithm 3.

4.3.3 Learning in observation space

We want to learn a visual policy π for assembling objects from diverse sets of primitives by a real robot. The sole input of the policy is the camera observation of the scene. We learn the image-action association with a supervised learning approach, Behavior Cloning (BC) [Pomerleau, 1989], where we obtain supervision with solutions found in the state space. Given the dataset $\mathcal{D}_\mu$ with state-actions pairs $\{(s_i, \boldsymbol{a}_i)\}$, we use a `pybullet` [Coumans, 2009] graphics renderer R to generate an RGB-D image $o_i = R(s_i)$ for each state s_i in $\mathcal{D}_\mu$. In order to allow multiple actions for each observation o_i, we generate an action-heatmap $h_i \in \mathcal{H}$ given the list of actions $\boldsymbol{a}_i = \{a_i^1, \ldots, a_i^l\}$. We record the observation-heatmap pairs to the dataset $\mathcal{D}_\pi$. The policy $\pi : \mathcal{O} \to \mathcal{H}$ is implemented as a CNN and is trained to predict correct action-heatmaps $\pi(o_i) = h_i$ for all $(o_i, h_i) \in \mathcal{D}_\pi$. We show an advantage of the heatmaps-based architecture over a network that directly predicts positions and orientations in Section 4.4.4.

For the task of building objects with a real robot, we consider separate pick and place actions that are parameterized by positions and orientations of primitives on the 2D plane of a table. For simplicity, we assume that the elevation of a primitive above the table can be estimated by external means such as an overhead depth camera or a force-feedback sensor of the robot arm. We define the output of our policy by distributions over 2D positions on the table plane and 3 possible orientations of primitives on the table. We represent such distributions as heatmaps $(h^{\text{pick}}, h^{\text{place}})$ corresponding to the source and target parameters of primitives. Our heatmaps are 4-channel images with one channel representing (x, y) position distribution of a pick or place action and three other channels representing orientation. To obtain the action parameters, we find the (x, y) coordinates as a maximum over the first heatmap channel. Given the extracted x and y location, we find the maximum over the 3 remaining channels at (x, y) to choose one out of 3 possible orientations. The placing positions and orientations might depend on the picked primitive, hence, we predict pick and place action-heatmaps sequentially.

We render separate observations $(o^{\text{pick}}, o^{\text{place}})$ for pick and place action-heatmaps $(h^{\text{pick}}, h^{\text{place}})$ respectively. For each list of pick-place actions $\boldsymbol{a}_i$, we render the observation $o_i^{\text{pick}} = R(s_i)$ and define h_i^{pick} as an image with Gaussian distributions around positions of all picked objects by $\boldsymbol{a}_i$. For each picked object, we render the observation $o_i^{\text{place}} = R(s_i)$ with the robotic arm

Algorithm 3 Pseudo-code of our approach.

Input: instance $\hat{s}$, classifier $f_{\mathcal{C}}$, simulator T, augmentation f_{sim2real}

function UNMAKE(state s, simulator T)
    $\mathcal{D} = \{s : (\text{value} = 1, \text{actions} = \{\})\}$
    for $s \in \mathcal{D}$ **do**
        *** Disassemble by moving non-blocked objects ***
        for $\tilde{a} \in$ DISASSEMBLY_ACTIONS(s) **do**
            $\tilde{s} = T(s, \tilde{a})$
            $\tilde{V} = \mathcal{D}[s].\text{value} * \gamma$ ▷ upper bound for the value
            $\tilde{\mathcal{D}} = \{\tilde{s} : (\text{value} = \tilde{V}, \text{actions} = \{\tilde{a}^{-1}\})\}$
            $\mathcal{D} =$ MERGE_SIMILAR_STATES($\mathcal{D}, \tilde{\mathcal{D}}$)
            if $\tilde{s}$.is_initial **then return** $\mathcal{D}$

*** Train the value function V ***
$\mathcal{C}_0^V = \{\hat{s}\}$ ▷ Known instances of the category
for $k \in \{0, \ldots, K-1\}$ **do** ▷ Number of value function iterations
    $\mathcal{D}_V^k = \{\}$ ▷ State-value pairs dataset
    for $s_i \in \mathcal{C}_k^V$ **do**
        $\mathcal{D}^i_{\text{unmake}} =$ UNMAKE(s_i)
        $\mathcal{D}^i_{\text{unmake}} =$ EXPAND_WITH_RANDOM_ACTIONS($\mathcal{D}^i_{\text{unmake}}$)
        $\mathcal{D}_V^k =$ MERGE_SIMILAR_STATES($\mathcal{D}_V^k, \mathcal{D}^i_{\text{unmake}}$)
    $V_k =$ TRAIN($\mathcal{D}_V^k$, fully_connected)
    $\mu_k =$ GREEDY_POLICY(V_k)
    $\mathcal{C}_{k+1}^V = \{\}$ ▷ Set of instances discovered with μ_k
    for $i \in \{1, \ldots,$ building_attempts$\}$ **do**
        $s_i =$ BUILD_INSTANCE(T, μ_k)
        if $f_{\mathcal{C}}(s_i) = 1$ **then**
            $\mathcal{C}_{k+1}^V$.append(s_i)

*** Train the visual policy π ***
$\mathcal{D}_\mu = \{\}$ ▷ Dataset of state-actions pairs
for $s_i \in \mathcal{C}_K^V$ **do**
    $\mathcal{D}^i_{\text{unmake}} =$ UNMAKE(s_i)
    $\mathcal{D}_\mu =$ MERGE_DATASETS($\mathcal{D}_\mu, \mathcal{D}^i_{\text{unmake}}$)
$\mathcal{D}_\pi = \{\}$ ▷ Dataset of observation-heatmap pairs
for $(s_i, \boldsymbol{a}_i) \in \mathcal{D}_\mu$ **do**
    $o_i^{\text{pick}}, o_i^{\text{place}} =$ RENDER(s_i)
    $h_i^{\text{pick}}, h_i^{\text{place}} =$ GENERATE_HEATMAPS($\boldsymbol{a}_i$)
    $\mathcal{D}_\pi$.append $\left((o_i^{\text{pick}}, h_i^{\text{pick}}), (o_i^{\text{place}}, h_i^{\text{place}})\right)$
$\pi =$ TRAIN($\mathcal{D}_\pi$, hourglass, f_{sim2real})
Output: assembly policy π for a real robot

picking this object and create the heatmap h_i^{place} with all possible placing positions in $\boldsymbol{a}_i$ (see Figure 4.4). We record all observation-heatmap pairs in the dataset $\mathcal{D}_\pi$.

Heatmaps are often used as CNN outputs in tasks such as human pose estimation [Newell et al., 2016, Wei et al., 2016] and segmentation [Shelhamer et al., 2017]. We follow [Newell et al., 2016] and use a HourGlass CNN architecture for heatmap prediction $\pi(o) \mapsto h$. Given $\mathcal{D}_\pi$, we train π by minimizing the loss $\hat{\theta} = \arg\min_\theta \text{MSE}(\pi_\theta(o_i), h_i)$, where $\hat{\theta}$ are parameters of the HourGlass CNN. We obtain parameters for the pick and place actions by maximizing the obtained location heatmaps over the 2D space and then maximizing orientations over the 3 channels at the selected location. The overview of our policy learning is illustrated by Figure 4.2c and lines 30-41 of Algorithm 3.

4.4 Results

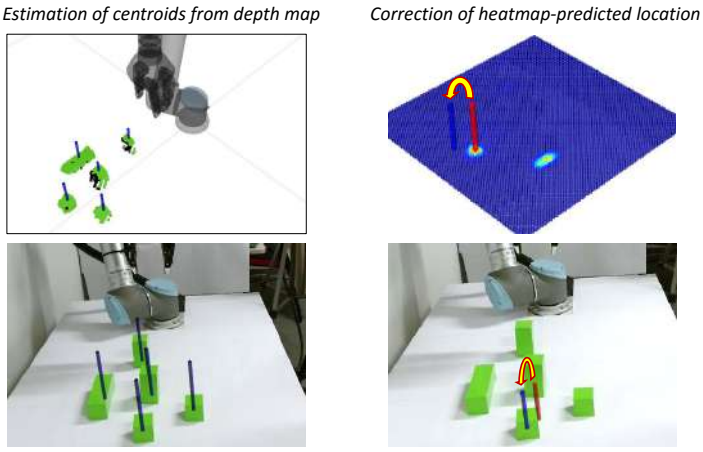


Figure 4.5 – Correction procedure applied to the spatial maxima of predicted heatmaps. Left: object centroids (blue bars) are estimated by the spatial clustering of depth-map locations above the table surface. Right: The location of the heatmap maxima (red bar) is corrected to the location of the closest object centroid. Refer to Section 4.4.1 for further explanations.

In this section we evaluate our approach both in simulation and on a real UR5 robot. We start with implementation details in Section 4.4.1 and present tasks used for evaluation in Section 4.4.2. Section 4.4.3 confirms the importance of learning the state-value function for efficient task solving.

Section 4.4.4 evaluates visual policies trained to solve tasks in the observation space. We validate our proposed network architecture and highlight the importance of the disassembling procedure. We show a few examples of successful and failed trails on a real robot in Section 4.4.5. Additional qualitative results are available on the project website [Pashevich et al., 2020b].

4.4.1 Implementation details

We control a 6-DoF UR5 robotic arm with a 3 finger Robotiq gripper. In simulation, we model the robot and its environment with the `pybullet` physics simulator [Coumans, 2009]. Given the positions and orientations of primitives to be manipulated, we use standard path planing methods [LaValle, 2006, Şucan et al., 2012] to implement the pick and place actions. The elevation of primitives above the table surface is obtained with a Microsoft Kinect-2 camera located above the table.

Our observations are depth images recorded with another Kinect-2 camera placed in front of the robot arm. We use the same parameters for real and simulation cameras (location and calibration). Visual policies receive the depth image and color segmentation masks corresponding to the colors of primitives. While visual policies in simulation have average errors of less than 5mm, the sim2real gap increases this value up to 2-3 cm on the real robot. Given that stacking multiple primitives requires high precision, we apply a correction procedure using the depth camera. As illustrated in Figure 4.5, given a depth map, we first obtain object centroids by clustering points above the table surface. Next, we predict the spatial location for the next action by maximizing the heatmap produced by the policy. The predicted location is then corrected to the location of the nearest cluster centroid. The applied corrections are typically in the order of 2-3 centimeters.

The neural network V_η for Value Function has five fully-connected layers with 128 hidden units, ReLU activations and Batch Normalization [Ioffe and Szegedy, 2015]. We train V_η for 20 iterations of 30 epochs each using Adam and LR=1e-3. The CNN π_θ contains one HourGlass [Newell et al., 2016] module which we compare to ResNet [He et al., 2016] in Section 4.4.4. The spatial dimensions of HourGlass input and output are 256x256 and 64x64 pixels respectively. We train π_θ using Adam and LR=2.5e-4 for 50 epochs. For both value and visual networks, we use datasets of size 200.000 value-state and heatmap-observation pairs correspondingly. To enable the transfer of policies to the real robot, we use the sim2real transfer technique [Pashevich et al., 2019a] to augment synthetic depth maps during

training. Color segmentation masks are augmented with Bernoulli noise. During rendering, we also randomize shapes of primitives by adding noise to spatial coordinates of points that define cube meshes. We use 500 episodes for evaluation in simulation and 20 trials on the real robot.

4.4.2 Tasks

Tower. The goal of the agent is to stack cubes in a specific order of colors (see Figure 4.6). In the beginning of the task, green, yellow and red cubes of size 1 unit (1U) are randomly distributed on the surface of a table. The unit corresponds to a physical size of 4.5 cm. The lowest cube is always green, the rest of the tower is defined as alternating yellow and red cubes. We use the Tower task to compare HourGlass and ResNet architectures in Section 4.4.4.

Arch. The agent needs to use all primitives available on a table to build an arch (see Figure 4.7). The construction primitives are cubes of size 1U and beams of length 2U and 3U. The arch shape category is defined as two symmetrical pillars with a bar bridging them. For example, pillars of an arch could be constructed from a 3U beam, three cubes or one cube and a 2U beam. Initially, all primitives are randomly distributed on the surface of the table. The beams can have three axes-parallel orientations. The primitives can have any color that differ from the color of the table. The location of pillars on the table is pre-defined. There are 49, 16 and 4 instances for 5U, 4U and 3U arch shapes correspondingly. We use the Arch task to evaluate the generalization of our method to the shape category and to show advantages of unmaking procedure in Sections 4.4.3 and 4.4.4.

4.4.3 Learning in state space

This section evaluates how efficient our approach in generalizing to 3D shapes in the state space. For evaluation, we use a simulated Arch task. Our approach receives a single shape instance as an input and learns a state-value function by unmaking this instance. Given the trained value function and the simulator, we obtain the state policy by iterating over all possible actions and taking the one that maximizes the value function prediction. This state policy is then used to discover new object instances. We iteratively repeat the state-value network training after unmaking the discovered instances during 20 iterations. We compare our approach to MCTS [Coulom, 2006] and random exploration. Similarly to the state policy, both baselines choose an object for picking and its placing location

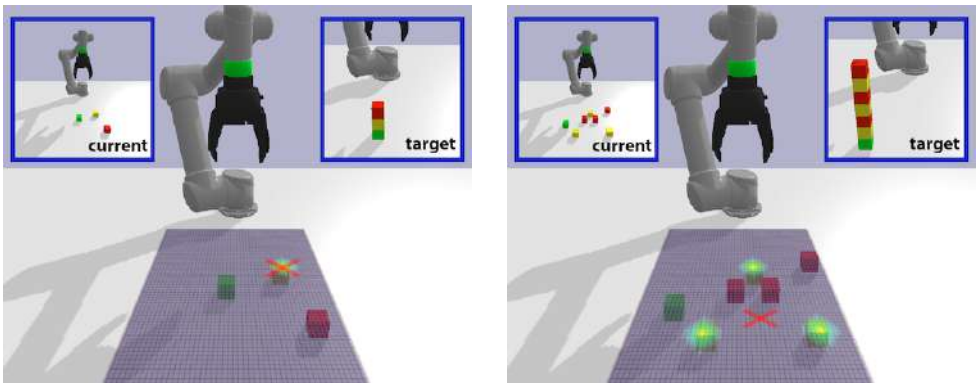


Figure 4.6 – Visualization of predictions of visual policies using HourGlass (green blobs) and ResNet-18 (red crosses) architectures. The policies are trained to build towers and should start by placing a yellow cube on the green one. ResNet predicts the correct picking location when all the cubes have distinct colors (left). Once identical yellow cubes are introduced to the scene (right), ResNet fails to choose between them and predicts an averaged location. HourGlass locates all the cubes correctly in both cases.

Arch, state space	3U	4U	5U
Random	$3.4e3 \pm 5.0e3$	$1.5e4 \pm 1.2e4$	$6.1e4 \pm 9.6e4$
MCTS [Coulom, 2006]	$6.3e2 \pm 4.7e2$	$7.0e3 \pm 6.2e3$	$1.6e4 \pm 2.4e4$
Ours (state policy)*	$\mathbf{4.0e0 \pm 0.7e0}$	$\mathbf{5.7e0 \pm 0.7e0}$	$\mathbf{6.9e0 \pm 1.8e0}$
Oracle	$4.0e0 \pm 0.7e0$	$5.7e0 \pm 0.7e0$	$6.4e0 \pm 1.0e0$

Table 4.1 – Average amount and standard deviation of steps required to build an arch shape for our method, random exploration and MCTS. *The simulation steps used for training of our method are not included.

on top of other objects. The baselines also choose the placing orientation among the three axis-parallel directions. For MCTS, we use a shape matching score which is defined by a percentage of how much the arch shape is completed with primitives.

We estimate an average amount of steps required by our method and the baselines to build an arch, and report results in Table 4.1. We also report the minimum number of steps required to build an arch (Oracle). Note that the Oracle has a non-zero variance since arches can be composed from different numbers of primitives. While the efficiency of our approach is comparable to Oracle, baselines require orders of magnitude more steps to find a correct solution. For example, to build a 5U arch, MCTS and

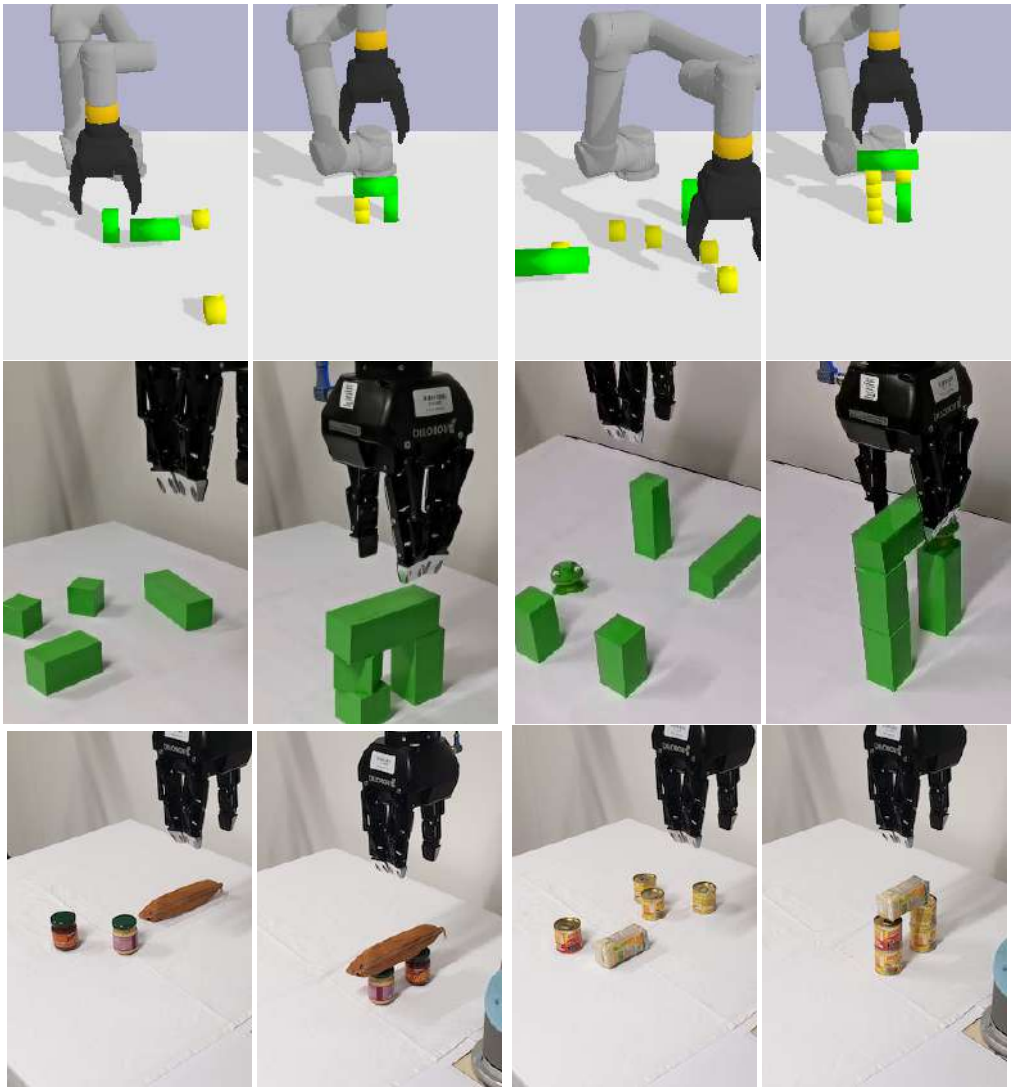


Figure 4.7 – Assembling arches of 3 and 5 units in simulation (top) and with a real robot (middle). Assembling arches with never-seen primitives resembling cubes and beams on the real robot (bottom). The image pairs correspond to initial and final states.

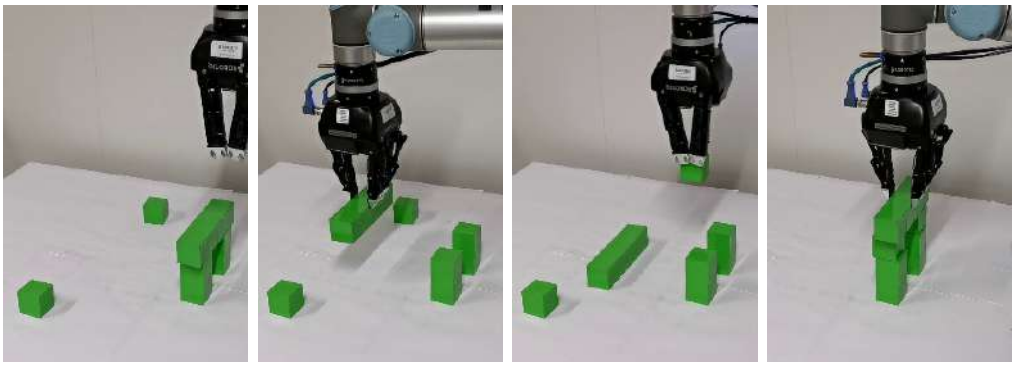


Figure 4.8 – Our visual policies successfully re-build shapes using primitives added to the table.

the random exploration require 1.6e4 and 6.1e4 steps respectively while our method solves the task in 6.9 steps on average. We expect our method to scale well to more complex tasks where the complexity of baselines will prohibit their use.

4.4.4 Evaluation of visual policies

This section evaluates visual policies trained to solve tasks given images as input. We validate the HourGlass architecture in Section 4.4.4, show benefits of the proposed unmaking approach over prior work in Section 4.4.4, evaluate generalization of our method to the shape category in Section 4.4.4 and present an evaluation on the real robot in Section 4.4.4.

Tower, simulation	3 cubes	5 cubes	7 cubes
ResNet-18 [He et al., 2016]	99.2	1.4	0.0
HourGlass [Newell et al., 2016]	**99.6**	**97.2**	**94.8**
Tower, real	3 cubes	5 cubes	7 cubes
HourGlass [Newell et al., 2016]	95.0	90.0	90.0

Table 4.2 – Success rates of visual policies (in percent) trained to build tower instances of 3, 5 and 7 cubes. On the real robot, the policies are evaluated using 20 trials.

Arch, simulation	3U	4U	5U
	category		
Single unmake trajectory	98.6	92.8	69.6
Multiple unmake trajectories	**99.0**	**98.8**	**95.6**

Table 4.3 – Success rates of visual policies trained to build the arch shape category of different heights. The policies are trained on trajectories obtained by disassembling the same object once or multiple times.

Visual policies architecture

We compare our heatmap-based architecture to a ResNet-18 network that directly outputs source and target parameters of manipulated primitives. Our approach uses HourGlass [Newell et al., 2016] to predict a multimodal distribution of picking and placing actions. The first heatmap corresponds to 2D locations on the robot workspace, the three additional heatmaps encode orientations of a primitive. ResNet-18 [He et al., 2016] predicts five values corresponding to 2D locations and three orientations of objects.

We train HourGlass and ResNet-18 networks to build towers of 3, 5 and 7 cubes of green, yellow and red colors. For ResNet-18, we adapt the learning rate to 1e-3 and the input image size to 224. Table 4.2 indicates that both HourGlass and ResNet policies achieve almost perfect accuracy for towers of 3 cubes (Figure 4.6, left). Given more complex scenes with multiple primitives of the same color, ResNet fails due to its unimodal prediction. As illustrated in Figure 4.6, it outputs a mean location of relevant primitives. The HourGlass-based policy builds towers of 7 cubes with a failure rate of 6% where the errors are mainly caused by occlusions. On the real-world Tower task with 7 cubes, the HourGlass-based policy succeeds in 18 trials out of 20. The two failure cases were caused by an occlusion and misidentifying a cube due to the sim2real gap. Empirically, we did not find any performance improvement when using multiple HourGlass modules.

Learning by unmaking

This section compares the proposed approach of unmaking assembled objects with prior work [Zakka et al., 2019]. While our method disassembles objects in multiple ways, [Zakka et al., 2019] proposes to use a single disassembly trajectory. Such disassembly trajectories consist of pairs of state and corresponding action. However, there could be multiple correct actions possible in a state as shown in Figure 4.4. We address this by computing

Arch, simulation	3U	4U	5U	3U	4U	5U
	instance			*category*		
Instance policies	99.4	97.8	96.8	61.4	54.4	32.6
Uni-height policies	**99.6**	**98.2**	**98.0**	**99.0**	**98.8**	**95.6**
Multi-height policy	97.4	96.4	95.4	97.8	95.4	94.0

Table 4.4 – Success rates of visual policies trained by disassembling only the input instance (top) and instances found by state policies. The state policies are trained on arches of the same height (middle) and arches of heights 3-5U (bottom). The policies assemble arches given a fixed set of primitives (left) or various configurations of primitives (right). While instance and uni-height policies need to be trained for each given arch height, a single multi-height policy can assemble arches of various heights.

several disassembly paths and merging actions that correspond to states, where the only difference comes from positions of primitives located on the table surface.

We train visual policies on observation-heatmap pairs obtained with and without multiple unmake trajectories. Table 4.3 shows that using multiple unmake paths significantly improves the performance. The performance difference is 26% on the hardest task of building 5U arches. Using multiple unmake paths make the policy learn multiple hypothesis of picking and placing locations. This property becomes critical when multiple identical objects are used or if there exist several shape instances where identical primitives are assembled in different configurations.

Learning to build a 3D category shapes

This section evaluates the generalization of our approach to a shape category. We train visual policies on trajectories of unmaking 3 sets of arch instances: (i) only the input instance, (ii) instances obtained by a state policy trained on arches of the same height, (iii) instances obtained by a state policy trained on arches of heights 3-5U. We refer to these policies as (i) instance, (ii) uni-height and (iii) multi-height. For instance and uni-height policies, we train separate networks to build arches of each height. For multi-height policy, the same network is used to build arches of varying heights.

We evaluate all the policies separately on the set of primitives corresponding to the input instance (Table 4.4, left) and on different sets corresponding to the entire category (Table 4.4, right). In the first case, all the

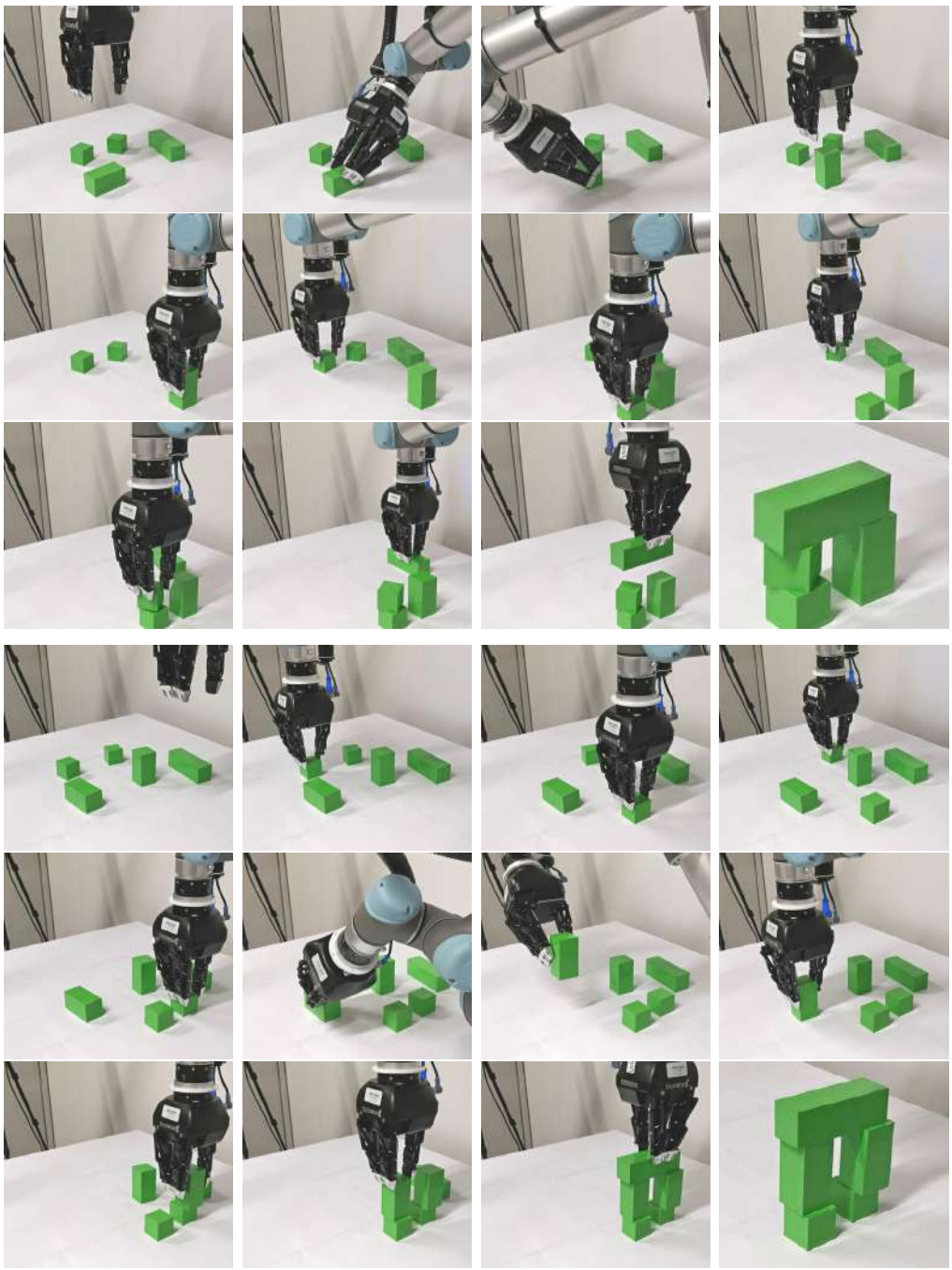


Figure 4.9 – Successful construction of an arch of 3 and 4 units from blocks with a real robot. Note the rotation movements performed by the robot to change the orientation of certain blocks.

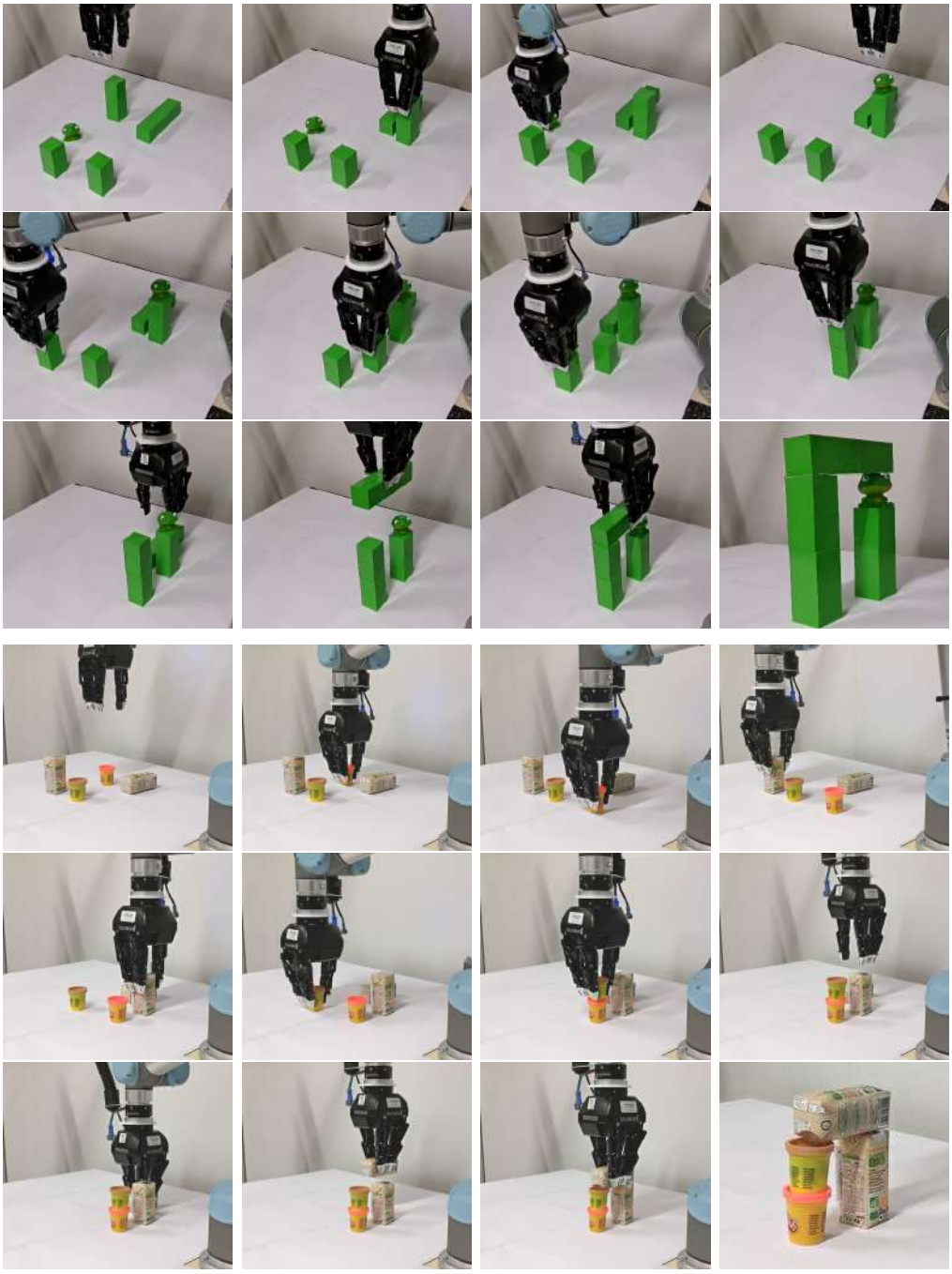


Figure 4.10 – Successful construction of an arch of 5 units from blocks with a real robot (top). Construction of arches from new primitives that have not been observed during training (bottom).

policies assemble arches with less than 5% of failures. The results of uni–height policies are higher compared to the instance policies. We believe that this improvement is due to a higher variation of the category instances that can be seen as a form of data augmentation. However, the performance of the instance policies rapidly drops when they are exposed to unseen sets of primitives. The uni-height policies have 2-3% higher success rates than the multi-height policy. Their success rates on 5U arches are 95.6% and 94.0% correspondingly. However, we need to train only a single network for the multi-height policy which is then able to reason about available primitives on the table and decide which arch height to build.

Real robot evaluation

This section evaluates the performance of our method on the real-world Arch task using two scenarios: (i) "normal" scenario that matches the simulation, (ii) "re-assembling" scenario. In the "normal" scenario the robot has to assemble arches from varying sets of primitives. Table 4.5 (left) shows that uni-height policies have similar performance to the multi-height policy in the standard scenario except for 3U arches. In the "re-assembling" scenario the task starts with an assembled arch and additional primitives on the table (see Figure 4.8). The agent is expected to re-assemble the arch by using all available primitives on the table. Our method is able to automatically re-assemble shapes without additional training due to the presence of random pick-place actions in the train set. Similarly to the value function dataset described in Section 4.3.2, we record observation-actions pairs corresponding to the inverse of the one-step random actions that include examples of re-assembling. In the "re-assembling" scenario (Table 4.5 (right)) the multi-height policy has higher success rates for 4U and 5U arches. We illustrate the assembly of arches with the real UR5 robot arm in Figures 4.7, 4.8 and on the project website [Pashevich et al., 2020b]. Failure cases with incorrect assembly are typically caused by occlusions and the gap between simulated and real environments.

Finally, we test the generalization of our approach to new building primitives and compare it to the MCTS baseline. We use three different sets of primitives that resemble cuboids used during training. The sets contain (i) 2 jars and a pencil case (Figure 4.7 bottom left), (ii) 4 cans and a juice box (Figure 4.7 bottom right), (iii) 5 stones (Figure 4.1 bottom). The input we provide to the MCTS baseline is the state of primitives in terms of their sizes and locations. Size and location are determined by clustering 3D points, which are obtained based on depth image coordinates above the table similarly to the prediction correction procedure used for our method.

Figure 4.11 – Construction of arches from new primitives that have not been observed during training. Despite the fact that our visual policy has been trained only with block-shaped primitives and only in simulation, it is able to generalize to new diverse primitives in a real robot setup.

Figure 4.12 – Failure cases for arch construction with a real robot. The failures originate from wrong choice of the target location (top example) and a failure to grasp a soft object (bottom example).

Arch, real	3U	4U	5U	3U	4U	5U
	normal scenario			*re-assembling*		
Uni-height policies	**95.0**	**80.0**	**75.0**	**90.0**	75.0	70.0
Multi-height policy	85.0	**80.0**	**75.0**	**90.0**	**80.0**	**80.0**

Table 4.5 – Success rates of visual policies assembling arches of different heights on a real robot. Over 20 trials, the policies are evaluated on a normal Arch task and a "re-assembling" scenario when the agent needs to adapt to additional primitives introduced to the scene after the arch had already been built.

Arch, real	set 1	set 2	set 3
	novel primitives		
Detect + MCTS [Coulom, 2006]	60.0	35.0	20.0
Our method	**90.0**	**80.0**	**55.0**

Table 4.6 – Success rates of our method and an MCTS baseline for assembling arches using novel primitives on a real robot. The arches built with primitives from sets 1, 2 and 3 are shown in Figure 4.7 (bottom left), Figure 4.7 (bottom right) and Figure 4.1 (bottom) correspondingly.

We estimate the spatial dimensions of primitives by fitting bounding boxes to depth points associated to each 3D cluster. Table 4.6 shows that our method significantly outperforms the MCTS baseline with the performance gap of up to 45%. In all cases the failures of MCTS are caused by errors in the estimation of size and location of the primitives. Given the incorrect estimates, MCTS cannot find an assembly path to build a correct arch. In contrast, our method relies on position correction based on direct image input and does not require spatial estimation of the primitives. Both methods often fail on the third set of primitives with stones given the substantial difference of stones to cuboid primitives used during training.

4.4.5 Qualitative results

Figures 4.9-4.12 demonstrate application of our method to the construction of arches on a real robot. Figures 4.9 and 4.10(top) present the construction of arches from blocks. The blocks are initially arranged in random positions and orientations. The policy changes positions and orientations of blocks, leading to the successful construction of arches of different sizes. Figures 4.10(bottom) and 4.11 demonstrate arch constructions from new

primitives that have not been observed during training. We emphasize that our learned visual policies directly control the robot without intermediate geometric reconstruction of the world state. Nevertheless, the policy can handle diverse object shapes. The robustness to new shapes could be further improved by using various shapes for policy training in simulation. Finally, Figure 4.12 illustrates few failure cases originating from incorrectly estimated pick and place locations as well as from a failure to grasp a soft object (a shoe).

4.5 Conclusion

In this chapter, we propose an approach to build 3D object shapes using a robotic arm and varying sets of primitive building blocks. Our method consists of two phases where the first phase solves a simpler task in the low-dimensional state space. The second phase uses the obtained solutions as supervision to train visual policies in the high-dimensional observation space. The state-space solutions are transferred to the observation space using a renderer of a simulator and heatmaps that represent all valid actions from a given state. We also propose a novel disassembly procedure that samples disassembly paths and reverses their actions to collect assembly demonstrations. We use our prior work described in Chapter 2 and train real-world policies using only synthetic images. We demonstrate successful application of our method to tasks of assembling tower and arch shapes on a real robot and show the ability of our policies to react on dynamic changes in the scene. Notably, our method can re-assemble shapes using additional building blocks introduced to the real scene and shows robust performance with unseen primitives resembling building blocks used during training.

5

Episodic Transformer for vision-and-language navigation

5.1 Introduction

Having an autonomous agent that can perform various household tasks given natural language commands is a long standing goal of the research community. To benchmark research progress, several simulated environments [Anderson et al., 2018, Shridhar et al., 2020, Puig et al., 2018] have recently emerged where the agents navigate and interact with the environment following natural language instructions. Solving the vision-and-language navigation (VLN) task requires the agent to ground human instructions in its embodied perception and action space. In practice, the agent is often required to perform long compositional tasks while observing only a small fraction of the environment from an egocentric point of view. Demonstrations manually annotated with human instructions are commonly used to teach an agent to accomplish specified tasks.

This chapter attempts to address two main challenges of VLN: (1) handling highly compositional tasks consisting of many subtasks and actions; (2) understanding the complex human instructions that are used to specify a task. Figure 5.1 shows an example task that illustrates both challenges. We show six key steps from a demonstration of 53 actions. To fulfill the task, the agent is expected to remember the location of a *fireplace* at $t = 0$ and use this knowledge much later (at $t = 31$). It also needs to solve object- (e.g."another vase") and location-grounded (e.g."where you were standing previously") coreference resolution in order to understand the human instructions.

Addressing the first challenge requires the agent to remember its past actions and observations. Most recent VLN approaches rely on recurrent ar-

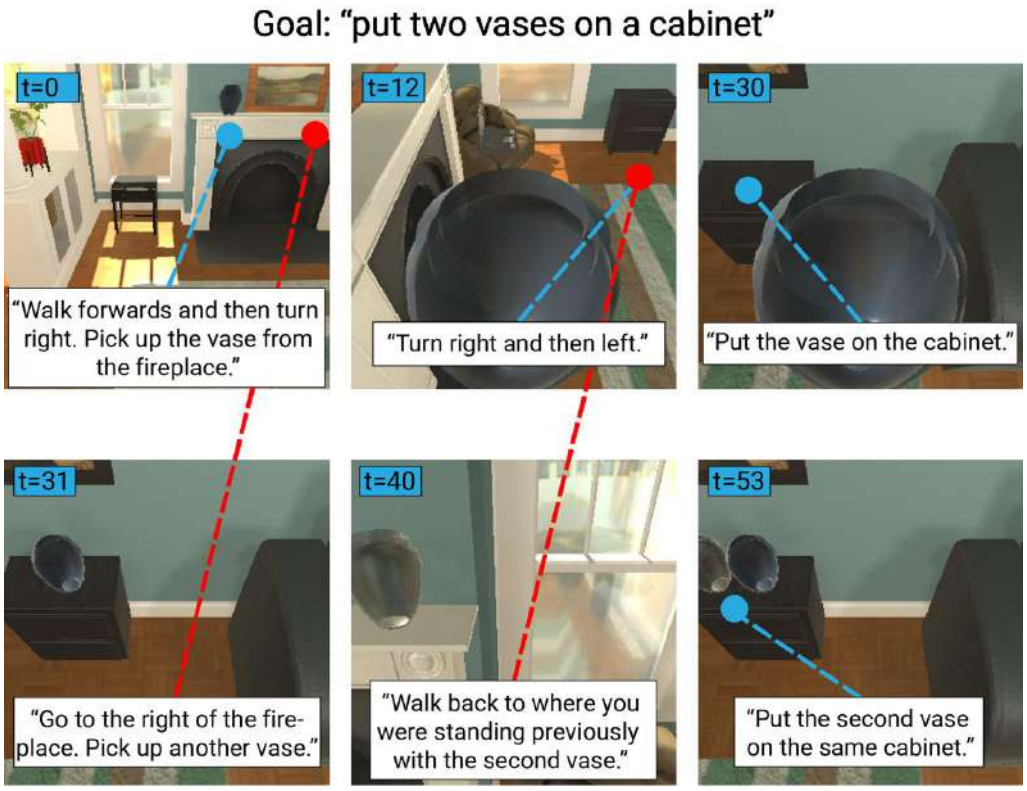


Figure 5.1 – An example of a compositional task in the ALFRED dataset [Shridhar et al., 2020] where the agent is asked to bring two vases to a cabinet. We show several frames from an expert demonstration with corresponding step-by-step instructions. The instructions expect the agent to be able to navigate to a *fireplace* which is not visible in its current egocentric view and to remember its previous location by referring to it as *"where you were standing previously"*.

chitectures [Tan et al., 2019, Wang et al., 2019b, Zhu et al., 2020, Ma et al., 2019a] where the internal state is expected to keep information about previous actions and observations. However, the recurrent networks are known to be inefficient in capturing long term dependencies [Vinyals et al., 2015] and may fail to execute long action sequences [Graves et al., 2016, Shridhar et al., 2020]. Motivated by the success of the attention-based transformer architecture [Vaswani et al., 2017] at language understanding [Devlin et al., 2019, Brown et al., 2020] and multimodal learning [Sun et al., 2019, Ding et al., 2020], we propose to use a transformer encoder to combine multimodal inputs including camera observations, language instructions, and previous actions. The transformer encoder has access to the history of the

entire episode to allow long-term memory and outputs the action to take next. We name our proposed architecture Episodic Transformer (E.T.).

Addressing the second challenge requires revisiting different ways to specify a task for the autonomous agent. We observe that domain-specific language [Ghallab et al., 1998], and temporal logic [Gopalan et al., 2018, Manna and Pnueli, 1992] can unambiguously specify the target states and (optionally) their temporal dependencies, while being decoupled from the visual appearance of a certain environment and the variations of human instructions. We hypothesize that using these *synthetic instructions* as an intermediate interface between the human and the agent would help the model to learn more easily and generalize better. To this end, we propose to pretrain the transformer-based language encoder in E.T. by predicting the synthetic instructions from human instructions. We also explore joint training, where human instructions and synthetic instructions are mapped into a shared latent space.

To evaluate the performance of E.T., we use the ALFRED dataset [Shridhar et al., 2020] which consists of longer episodes than the other vision-and-language navigation datasets [Anderson et al., 2018, Puig et al., 2018, Chen et al., 2019a], and also requires object interaction. We experimentally show that E.T. benefits from full episode memory and is better at solving tasks with long horizons than recurrent models. We also observe significant gains by pretraining the language encoder with the synthetic instructions. Furthermore, we show that when used for training jointly with natural language such intermediate representations outperform conventional data augmentation techniques for vision-and-language navigation [Fried et al., 2018] and work better than image-based annotations [Lynch et al., 2019].

In summary, our two main contributions are as follows. First, we propose Episodic Transformer (E.T.), an attention-based architecture for vision-and-language navigation and demonstrate its advantages over recurrent models. Second, we propose to use synthetic instructions as the intermediate interface between the human and the agent. Both contributions combined allow us to achieve a new state-of-the-art on the challenging ALFRED dataset. Our code and models are available on the project page [Pashevich et al., 2021b].

5.2 Related work

Instruction following agents. Building systems to understand and execute human instructions has been the subject of many previous work [Bugmann et al., 2001, Branavan et al., 2009, Tenorth et al., 2010, MacMahon

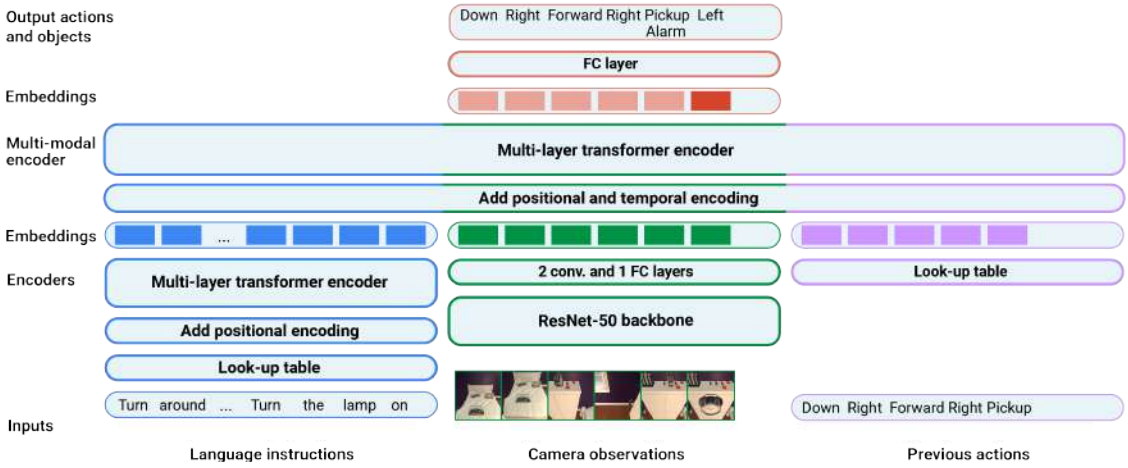


Figure 5.2 – Episodic Transformer (E.T.) architecture. To predict the next action, the E.T. model is given a natural language instruction $x_{1:L}$, visual observations since the beginning of an episode $v_{1:t}$ and previously taken actions $a_{1:t-1}$. Here we show an example that corresponds to the 6^{th} timestep of an episode: $t = 6$. After processing $x_{1:L}$ with a transformer-based language encoder, embedding $v_{1:t}$ with a ResNet-50 backbone and passing $a_{1:t-1}$ through a look-up table, the agent outputs t actions. During training we use all predicted actions for a gradient descent step. At test time, we apply the last action a_t to the environment.

et al., 2006, Chen and Mooney, 2011, Bollini et al., 2013, Misra et al., 2017, Misra et al., 2016, Paul et al., 2018, Lynch et al., 2019]. Instruction types include structured commands or logic programs [Manna and Pnueli, 1992, Ghallab et al., 1998, Puig et al., 2018], natural language [Chen and Mooney, 2011, Tellex et al., 2011], target state images [Lynch et al., 2019] or a mix [Lynch and Sermanet, 2020]. While earlier work focuses on mapping instructions and structured world states into actions [Mei et al., 2016, Andreas and Klein, 2015, Patel et al., 2019], it is desirable for the agents to be able to handle raw sensory inputs, such as images or videos. To address this, the visual-and-language navigation (VLN) task is proposed to introduce rich, unstructured visual context for the agent to explore, perceive and execute upon [Anderson et al., 2018, Ku et al., 2020, Chen et al., 2019a, Mehta et al., 2020, Krantz et al., 2020]. The agent is requested to navigate to the target location based on human instructions and real, or photo-realistic image inputs, implemented as navigation graphs [Anderson et al., 2018, Chen et al., 2019a] or continuous environment [Krantz et al., 2020] in simulators [Todorov et al., 2012, Kolve et al., 2017, Deitke et al.,

2020, Savva et al., 2019]. More recently, the ALFRED environment [Shridhar et al., 2020] introduces the object interaction component to complement visual-language navigation. It is a more challenging setup as sequences are longer than in other vision-language navigation datasets and all steps of a sequence have to be executed properly to succeed. We focus on the ALFRED environment and its defined tasks.

Training a neural agent for VLN. State-of-the-art models in language grounded navigation are neural agents trained using either Imitation Learning [Fried et al., 2018], Reinforcement Learning [Li et al., 2020] or a combination of both [Tan et al., 2019, Wang et al., 2019b]. In addition, auxiliary tasks, such as progress estimation [Ma et al., 2019b, Ma et al., 2019a], back-tracking [Ke et al., 2019], speaker-driven route selection [Fried et al., 2018], cross-modal matching [Wang et al., 2019b, Huang et al., 2019], back translation [Tan et al., 2019] and text-based agent training [Côté et al., 2018, Shridhar et al., 2021] are proposed to improve the performance and generalization of neural agents in seen and unseen environments. Most of these approaches use recurrent neural networks and encode previous observations and actions as hidden states. This work proposes to leverage transformers [Vaswani et al., 2017] which enables encoding the full episode of history for long-term navigation and interaction. Most relevant to our approach are VLN-BERT [Majumdar et al., 2020] and Recurrent VLBERT [Hong et al., 2021], which also employ transformers for VLN. Unlike our approach, VLN-BERT [Majumdar et al., 2020] trains a transformer to measure the compatibility of an instruction and a set of already generated trajectories. The concurrent work Recurrent VLBERT [Hong et al., 2021] proposes to use an explicit recurrent state and a pretrained VLBERT to process one observation for each timestep. This shortens the model's memory and might be harmful for long horizon tasks such as ALFRED. In contrast, we do not introduce any recurrency and process the whole history of observations at once.

Multimodal Transformers. Transformers [Vaswani et al., 2017] have brought success to a wide range of classification and generation tasks, from language [Vaswani et al., 2017, Devlin et al., 2019, Brown et al., 2020], to images [Dosovitskiy et al., 2021, Carion et al., 2020] and videos [Girdhar et al., 2019, Wang et al., 2018]. In [Parisotto et al., 2020], the authors show that training transformers for long time horizon planning with RL is challenging and propose a solution. The convergence of the transformer architecture for different problem domains also leads to multimodal trans-

formers, where a unified transformer model is tasked to solve problems that require multimodal information, such as visual question answering [Lu et al., 2019], video captioning and temporal prediction [Sun et al., 2019], or retrieval [Gabeur et al., 2020]. Our Episodic Transformer can be considered a multimodal transformers, where the inputs are language (instructions), vision (images) and actions.

Semantic parsing of human instructions. Semantic parsing focuses on converting natural language into logic forms that can be interpreted by machines. It has applications in question answering [Zelle and Mooney, 1996, Zettlemoyer and Collins, 2007, Berant et al., 2013], and can be learned either with paired supervision [Zettlemoyer and Collins, 2005, Yu et al., 2018, Berant et al., 2013] or weak supervision [Artzi and Zettlemoyer, 2013, Patel et al., 2020]. For instruction following, semantic parsing has been applied to map natural language into lambda calculus expressions [Artzi and Zettlemoyer, 2013] or linear temporal logic [Patel et al., 2020]. We show that rather than directly using the semantic parsing outputs, it is more beneficial to transfer its pretrained language encoder to the downstream VLN task.

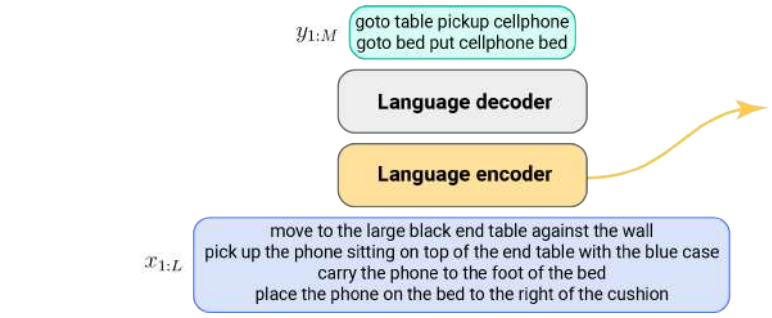
5.3 Method

We first define the vision-and-language navigation task in Section 5.3.1 and describe the Episodic Transformer (E.T.) model in Section 5.3.2. We then introduce the synthetic language and explain how we leverage it for pretraining and joint training in Section 5.3.3.

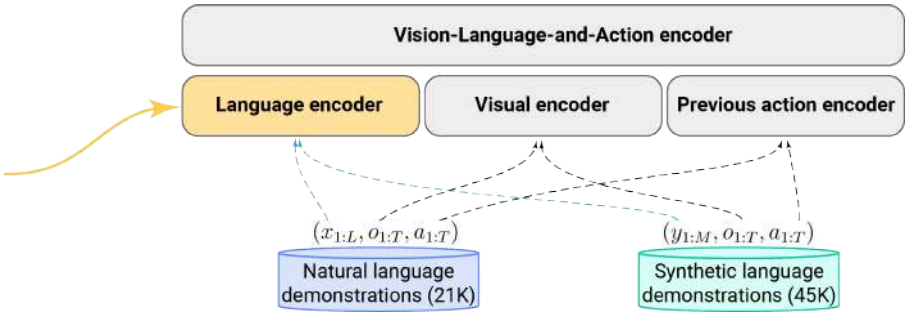
5.3.1 VLN background

The vision-and-language navigation task requires an agent to navigate in an environment and to reach a goal specified by a natural language instruction. Each demonstration is a tuple $(x_{1:L}, o_{1:T}, a_{1:T})$ of a natural language instruction, expert visual observations, and expert actions. The instruction $x_{1:L}$ is a sequence of L word tokens $x_i \in \mathbb{R}$. The visual observations $o_{1:T}$ is a sequence of T camera images $o_t \in \mathbb{R}^{W \times H \times 3}$ where T is the demonstration length and $W \times H$ is the image size. The expert actions $a_{1:T}$ is a sequence of T action type labels $a_t \in \{1, \ldots, A\}$ used by the expert and A is the number of action types.

The goal is to learn an agent function f which approximates the expert policy. In the case of a recurrent architecture, the agent predicts the next



(a) Pretraining on natural to synthetic translation.



(b) Joint training using natural and synthetic annotations.

Figure 5.3 – Training with natural and synthetic language. Top: We pretrain the language encoder of the model to translate natural language instructions to synthetic language instructions. Due to a more task-oriented synthetic representation, the language encoder learns a better representation. We use the language encoder weights to initialize the language encoder of the agent (shown in yellow). Bottom: We jointly use demonstrations annotated with natural language and demonstrations annotated with synthetic language to train the agent. Due to the larger size of the synthetic language dataset, the resulting agent has better performance even when evaluated on natural language annotations.

action $\hat{a}_t$ given a language instruction $x_{1:L}$, a visual observation o_t, the previously taken action $\hat{a}_{t-1}$, and uses its hidden state h_{t-1} to keep track of the history:

$$\hat{a}_t, h_t = f(x_{1:L}, o_t, \hat{a}_{t-1}, h_{t-1}). \tag{5.1}$$

For an agent with full episode observability, all previous visual observations $o_{1:t}$ and all previous actions $\hat{a}_{1:t-1}$ are provided to the agent directly and no hidden state is required:

$$\hat{a}_t = f(x_{1:L}, o_{1:t}, \hat{a}_{1:t-1}). \tag{5.2}$$

5.3.2 Episodic Transformer model

Our Episodic Transformer (E.T.) model shown in Figure 5.2 relies on attention-based multi-layer transformer encoders [Vaswani et al., 2017]. It has no hidden state and observes the full history of visual observations and previous actions. To inject information about the order of words, frames and action sequences, we apply the sinusoidal encoding to transformer inputs. We refer to this encoding as positional encoding for language tokens and temporal encoding for expert observations and actions.

Our E.T. architecture consists of four encoders: language encoder, visual encoder, action encoder, and multimodal encoder. The language encoder shown in the bottom left part of Figure 5.2 gets instruction tokens $x_{1:L}$ as input. It consists of a look-up table and a multi-layer transformer encoder and outputs a sequence of contextualized language embeddings $h^x_{1:L}$. The visual encoder shown in the bottom center part of Figure 5.2 is a ResNet-50 backbone [He et al., 2016] followed by 2 convolutional and 1 fully-connected layers. The visual encoder projects a visual observation o_t into its embedding h^o_t. All the episode visual observations $o_{1:T}$ are projected independently using the same encoder. The action encoder is a look-up table shown in the bottom right part of Figure 5.2 which maps action types $a_{1:T}$ into action embeddings $h^a_{1:T}$.

The multimodal encoder is a multi-layer transformer encoder shown in the middle of Figure 5.2. Given the concatenated embeddings from modality specific encoders $(h^x_{1:L}, h^o_{1:T}, h^a_{1:T})$, the multimodal encoder returns output embeddings $(z^x_{1:L}, z^o_{1:T}, z^a_{1:T})$. The multimodal encoder employs causal attention [Vaswani et al., 2017] to prevent visual and action embeddings from attending to subsequent timesteps. We take the output embeddings $z^o_{1:T}$ and add a single fully-connected layer to predict agent actions $\hat{a}_{1:T}$.

During E.T. training, we take advantage of the sequential nature of the transformer architecture. We input a language instruction $x_{1:L}$ as well as all visual observations $o_{1:T}$ and all actions $a_{1:T}$ of an expert demonstration to the model. The E.T. model predicts all actions $\hat{a}_{1:T}$ at once as shown at the top of Figure 5.2. We compute and minimize the cross-entropy loss between predicted actions $\hat{a}_{1:T}$ and expert actions $a_{1:T}$. During testing at timestep t, we input visual observations $o_{1:t}$ up to a current timestep and previous actions $\hat{a}_{1:t-1}$ taken by the agent. We select the action predicted for the last timestep $\hat{a}_t, \hat{c}_t$ and apply it to the environment which generates the next visual observation o_{t+1}. In Figure 5.2 we show an example that corresponds to the 6^{th} timestep of an episode where the action `Left` will be taken next.

5.3.3 Synthetic language

To improve understanding human instructions presenting a wide range of variability, we propose to pretrain the agent language encoder with a translation into a synthetic language, see Figure 5.3a. We also generate additional demonstrations, annotate them with synthetic language and jointly train the agent using both synthetic and natural language demonstrations, see Figure 5.3b.

An example of the synthetic language and a corresponding natural language instruction is shown in Figure 5.3a. The synthetic annotation is generated for each expert demonstration using the expert path planner arguments. In ALFRED, each expert path is defined with Planning Domain Definition Language (PDDL) [Ghallab et al., 1998] which consists of several subgoal actions. Each subgoal action has a type and a target class, e.g. `Put Apple Table` or `Goto Bed` which we use as a synthetic annotation for this subgoal action. Note that such annotation only defines a class but not an instance of the target. We annotate each expert demonstration with subgoal action annotations concatenated in chronological order to produce a synthetic annotation $y_{1:M}$.

We use synthetic language to pretrain the language encoder of the agent on a sequence-to-sequence (seq2seq) translation task. The translation dataset consists of corresponding pairs $(x_{1:L}, y_{1:M})$ of natural and synthetic instructions. The translation model consists of a language encoder and a language decoder as shown in Figure 5.3a. The language encoder is identical to the agent language encoder described in Section 5.3.2. The language decoder is a multi-layer transformer decoder with positional encoding and same hyperparameters as the encoder. Given a natural language annotation $x_{1:L}$, we use the language encoder to produce embeddings $h_{1:L}$. The embeddings are passed to the language decoder which predicts N translation tokens $\hat{y}_i$. We train the model by minimizing the cross-entropy loss between predictions $\hat{y}_{1:N}$ and synthetic annotations $y_{1:M}$. Once the training converges, we use the weights of the translator language encoder to initialize the language encoder of the agent.

We also explore joint training by generating an additional dataset of expert demonstrations annotated with synthetic language. We use the AI2-THOR simulator [Kolve et al., 2017] and scripts provided by Shridhar *et al.* [Shridhar et al., 2020]. Apart from the annotations, the synthetic dataset differs from the original one in terms of objects configurations and agent initial positions. We train the agent to predict actions using both natural language and synthetic language datasets as shown in Figure 5.3b. We use the same language, vision and action encoders for both datasets but

use two different look-up tables for natural and synthetic language tokens which we found to work the best experimentally. For both datasets, we sample batches of the same size, compute the loss for both batches and do a single gradient descent step. After a fixed number of training epochs, we evaluate the agent on natural and synthetic language separately using the same set of validation tasks.

5.4 Results

In this section, we ablate different components of E.T. and compare E.T. with state-of-the-art methods. First, we describe the experimental setup and the dataset in Section 5.4.1. Next, we compare our method to a recurrent baseline and highlight the importance of full episode observability in Section 5.4.2. We then study the impact of joint training and pretraining with synthetic instructions in Section 5.4.3 and compare with previous state-of-the-art methods on the ALFRED dataset in Section 5.4.4. Finally, we provide qualitative results including visualizations of transformer attention maps and examples of agent trajectories in Section 5.4.5.

5.4.1 Experimental setup

Dataset. The ALFRED dataset [Shridhar et al., 2020] consists of demonstrations of an agent performing household tasks following natural language defined goals. The tasks are compositional with nonreversible state changes. The dataset includes $8,055$ expert trajectories $(o_{1:T}, a_{1:T})$ annotated with $25,743$ natural language instructions $x_{1:L}$. It is split into $21,023$ train, $1,641$ validation and $3,062$ test annotations. The validation and test folds are divided into *seen* splits which contain environments from the train fold and *unseen* splits which contain new environments. To leverage synthetic instructions to pretrain a language encoder, we pair every annotated instruction $x_{1:L}$ with its corresponding synthetic instruction $y_{1:M}$ in the train fold. For joint training, we generate $44,996$ demonstrations $(y_{1:M}, o_{1:T}, a_{1:T})$ from the train environments annotated automatically with synthetic instructions. For ablation studies in Section 5.4.2 and Section 5.4.3, we use the validation folds only. For comparison with state-of-the-art in Section 5.4.4, we report results on both validation and test folds.

Baselines. In Section 5.4.2, we compare our model to a model based on a bi-directional LSTM [Shridhar et al., 2020]. We use the same hyperparameters as Shridhar *et al.* [Shridhar et al., 2020] and set the language encoder

hidden size to 100, the action decoder hidden size to 512, the visual embeddings size to 2500, and use 0.3 dropout for the decoder hidden state. We experimentally find the Adam optimizer with no weight decay and a weight coefficient 0.1 for the target class cross-entropy loss to work best. The LSTM model uses the same visual encoder as the E.T. model. In Section 5.4.4, we also compare our model to MOCA [Singh et al., 2020] and the model of Nguyen *et al.* [Van-Quang Nguyen, 2020].

Evaluation metrics. For our ablation studies in Sections 5.4.2 and 5.4.3, we report agent success rates. To understand the performance difference with recurrent-based architectures in Section 5.4.2, we also report success rates on individual subgoals. This metric corresponds to the proportion of subgoal tasks completed after following an expert demonstration until the beginning of the subgoal and conditioned on the entire language instruction. We note that the average task length is 50 timesteps while the average length of a subgoal is 7 timesteps.

Implementation details. Among the 13 possible action types, 7 actions involve interacting with a target object in the environment. The target object of an action a_t is chosen with a binary mask $m_t \in \{0,1\}^{W \times H}$ which specifies the pixels of visual observation o_t that belongs to the target object. There are 119 object classes in total. The pixel masks m_t are provided along with expert demonstrations during training. We follow Singh *et al.* [Singh et al., 2020] and ask our agent to predict the target object class c_t, which is then used to retrieve the corresponding pixel mask $\hat{m}_t$ generated by a pretrained instance segmentation model. The segmentation model takes o_t as input and outputs $(\hat{c}_t, \hat{m}_t)$.

The agent observations are resized to 224×224. The mask generator receives images of size 300×300 following Singh *et al.* [Singh et al., 2020]. Both the visual encoder and the mask generator are pretrained on a dataset of 325K frames of expert demonstrations from the train fold and corresponding class segmentation masks. We use ResNet-50 Faster R-CNN [Ren et al., 2015] for the visual encoder pretraining and ResNet-50 Mask R-CNN [He et al., 2017] for the mask generator. We do not update the mask generator and the visual encoder ResNet backbone during the agent training. In the visual encoder, ResNet features are average-pooled 4 times to reduce their size and 0.3 dropout is applied. Resulting feature maps of $512 \times 7 \times 7$ are fed into 2 convolutional layers with 256 and 64 filters of size 1 by 1 and mapped into an embedding of the size 768 with a fully connected layer.

Model	Task		Sub-goal	
	Seen	Unseen	Seen	Unseen
LSTM	23.2	2.4	75.5	58.7
LSTM + E.T. enc.	27.8	**3.3**	76.6	59.5
E.T.	**33.8**	3.2	**77.3**	**59.6**

Table 5.1 – Comparison of E.T. and LSTM architectures: (1) an LSTM-based model [Shridhar et al., 2020], (2) an LSTM-based model trained with the transformer language encoder of the E.T. model, (3) E.T., our transformer-based model. All models are trained using the natural language dataset only and evaluated on validation folds. The two parts of the table show the success rate for tasks (average length 50) and sub-goals (average length 7). While the sub-goal success rates of all models are relatively close, E.T. outperforms both recurrent agents on full tasks which highlights the importance of full episode observability.

Both transformer encoders of E.T. have 2 blocks, 12 self-attention heads and the hidden size of 768. We use 0.1 dropout inside transformer encoders.

We use the AdamW optimizer [Loshchilov and Hutter, 2019] with 0.33 weight decay and train the model for 20 epochs. Every epoch includes 3,750 batches of 8 demonstrations each. For joint training, each batch consists of 4 demonstrations with human instructions and 4 demonstrations with synthetic instructions. For all experiments, we use a learning rate of 10^{-4} during the first 10 epochs and 10^{-5} during the last 10 epochs. Following Shridhar *et al.* [Shridhar et al., 2020], we use auxiliary losses for overall and subgoal progress [Ma et al., 2019a] which we sum to the model cross-entropy loss with weights 0.1. All the hyperparameter choices were made using a moderate size grid search. Once the training is finished, we evaluate every 2-nd epoch on the validation folds. Following Singh *et al.* [Singh et al., 2020], we use Instance Association in Time and Obstruction Detection modules during evaluation.

5.4.2 Model analysis

Comparison with recurrent models. To validate the gain due to an episodic memory, we compare the E.T. architecture with a model based on a recurrent LSTM architecture. We train both models using the dataset with natural language annotations only. As shown in Table 5.1, the recurrent model succeeds in 23.2% of tasks in seen environments and in 2.4% of tasks in unseen environments. E.T. succeeds in 33.8% and 3.2% of tasks re-

Visible	Frames		Actions	
	Seen	Unseen	Seen	Unseen
None	0.5	0.2	23.7	1.7
1 last	28.9	2.2	**33.8**	**3.2**
4 last	31.5	2.0	32.0	2.4
16 last	33.5	2.9	31.1	2.8
All	**33.8**	**3.2**	27.1	2.2

Table 5.2 – Ablation on accessible history length of E.T., in terms of visual frames (left two columns) and actions (right two columns).

spectively which is a relative improvement of 45.6% and 33.3% compared to the LSTM-based agent. However, the success rate computed for individual subgoals shows only 2.3% and 1.5% of relative improvement of E.T. over the recurrent agent in seen and unseen environments respectively. We note that a task consists on average of 6.5 subgoals which makes the long-term memory much more important for solving full tasks.

To understand the performance difference, we train an LSTM-based model with the E.T. language encoder. Given that both LSTM and E.T. agents receive the same visual features processed by the frozen ResNet-50 backbone and have the same language encoder architecture, the principal difference between the two models is the processing of previous observations. While the E.T. agent observes all previous frames using the attention mechanism, the LSTM-based model relies on its recurrent state and explicitly observes only the last visual frame. The recurrent model performance shown in the 2-nd row of Table 5.1 is similar to the E.T. performance in unseen environments but is 17.7% less successful than E.T. in seen environments. This comparison highlights the importance of the attention mechanism and full episode observability. We note that E.T. needs only one forward pass for a gradient descent update on a full episode. In contrast, the LSTM models need to do a separate forward pass for each episode timestep which significantly increases their training time with respect to E.T. models. We further compare how E.T. and LSTM models scale with additional demonstrations in Section 5.4.3.

Accessible history length. We train E.T. using different lengths of the episode history observed by the agent in terms of visual frames and previous actions and show the results in Table 5.2. The first two columns of Table 5.2 compare different lengths of visual observations history from no past frames to the entire episode. The results indicate that having access

to all visual observations is important for the model performance. We note that performance of the model with 16 input frames is close to the performance of the full episode memory agent, which can be explained the average task length of 50 timesteps.

The last two columns of Table 5.2 show that the agent does not benefit from accessing more than one past action. This behavior can be explained by "causal misidentification" phenomenon: access to more information can yield worse performance [de Haan et al., 2019]. It can also be explained by poor generalizability due to overfitting of the model to expert demonstrations. We also note that the model observing no previous actions is 29.8% and 46.8% relatively less successful in seen and unseen environments than the agent observing the last action. We therefore fix the memory size to be unlimited for visual observations and to be 1 timestep for the previous actions.

Model capacity. Transformer-based models are known to be expressive but prone to overfitting. We study how the model capacity impacts the performance while training on the original ALFRED dataset. We change the number of transformer blocks in the language encoder and the multimodal encoder and report results in Table 5.3. The results indicate that the model with a single transformer block is not expressive enough and the models with 3 and 4 blocks overfit to the train data. The model with 2 blocks represents a trade-off between under- and overfitting and we therefore keep this value for all the experiments.

Attention visualization. We present visualizations of visual and text attention heatmaps in Section 5.4.5.

5.4.3 Training with synthetic annotations

Joint training. We train the E.T. model using the original dataset of 21,023 expert demonstrations annotated with natural language and the additionally generated dataset of 44,996 expert demonstrations with synthetic annotations. We compare three types of synthetic annotations: (1) direct use of visual embeddings from the expert demonstration frames, no language instruction is generated. A similar approach can be found in Lynch and Sermanet [Lynch and Sermanet, 2020]; (2) train a model to generate instructions, e.g.with a speaker model [Fried et al., 2018], where the inputs are visual embeddings from the expert demonstration frames, and the targets are human annotated instructions; and (3) subgoal actions and

# Blocks	Seen	Unseen
1	25.0	1.6
2	**33.8**	**3.2**
3	28.6	2.2
4	19.8	1.1

Table 5.3 – Ablation of E.T. model capacity. We compare E.T. models with different number of transformer blocks in language and multimodal encoders.

Synthetic instr.	Test on synthetic		Test on human	
	Seen	Unseen	Seen	Unseen
Expert frames	**54.0**	**6.1**	28.5	3.4
Speaker text	36.3	3.1	37.4	3.9
Subgoal actions	47.2	5.9	**38.5**	**5.4**
No synthetic	-	-	33.8	3.2

Table 5.4 – Comparison of different synthetic instructions used for **joint training**. We jointly train E.T. using demonstrations with human annotations and demonstrations with different types of synthetic instructions. In the first two columns, we evaluate the resulting models using the same type of synthetic annotations that is used during training. In the last two column, the models are evaluated on human annotated instructions.

objects annotations described in Section 5.3.3. For (1), we experimentally find using all expert frames from a demonstration works significantly better than a subset of frames. The visual embeddings used in (1) and (2) are extracted from a pretrained frozen ResNet-50 described in Section 5.4.1. To generate speaker annotations, we use a transformer-based seq2seq model (Section 5.3.3) with the difference that the inputs are visual embeddings instead of text.

We report success rates of models trained jointly and evaluated independently on synthetic and human annotated instructions in Table 5.4. The results are reported on the validation folds. The model trained on expert frames achieves the highest performance when evaluated on synthetic instructions. However, when evaluated on human instructions, this model has 15.6% relatively lower success rate in seen environments than the baseline without joint training. This indicates that the agent trained to take expert frames as instructions does not generalize well to human instructions. Using

Train data	Seen	Unseen
21K human only	33.8	3.2
21K human & 11K synth.	35.5	4.1
21K human & 22K synth.	38.3	**5.5**
21K human & 44K synth.	**38.5**	5.4

Table 5.5 – **Joint training** of the E.T. model using different number of demonstrations annotated with subgoal actions. We report success rates on the validation folds.

Train data	LSTM		E.T.	
	Seen	Unseen	Seen	Unseen
Human annotations	23.2	2.4	33.8	3.2
Human + synthetic	25.2	2.9	**38.5**	**5.4**

Table 5.6 – Comparison of an LSTM-based model and E.T. **trained jointly with demonstrations annotated by subgoal actions**. The results indicate that E.T. scales better with additional data than the LSTM-based agent.

speaker translation annotations improves over the no joint training baseline by 10.6% and 21.8% relative in seen and unseen environments respectively. Furthermore, our proposed subgoal annotations bring an even larger improvement of 13.9% and 68.7% relative in seen and unseen environments which highlights the benefits of joint training with synthetic instructions in the form of subgoal actions.

We further study the impact of additional data and train the E.T. agent using different number of demonstrations annotated with subgoal actions. The results are shown in Table 5.5. We can see that increasing the number of synthetic demonstrations in the joint training up to 22K brings a significant improvement over the model trained on human annotations only. However, doubling the synthetic demonstrations up to 44K has a very minor impact on the agent performance. We use 44K demonstrations with synthetic annotations for further experiments with joint training.

Finally, we study if the recurrent baseline also benefits from joint training with synthetic data. Table 5.6 shows that the relative gains of joint training are 2.3 and 4.4 times higher for E.T. than for the LSTM-based agent in seen and unseen environments respectively. These numbers clearly show that E.T. benefits more from additional data and confirms the advan-

Objective	Transfer	Seen	Unseen
None	-	33.8	3.2
BERT	Text embedding	32.3	3.4
Seq2seq	Translated text	35.2	3.6
Seq2seq	Text encoder	**37.6**	**3.8**

Table 5.7 – Comparison of models **with different language encoder pretraining** strategies. We pretrain a seq2seq model to map human instructions into synthetic instructions and transfer either its output text (third row) or its learned weights (fourth row). For completeness we also compare with no pretraining (first row) and BERT pretraining (second row).

tage of our model over LSTM-based agents.

Language encoder pretraining. Another application of synthetic instructions is to use them as an intermediate representation that decouples the visual appearance of an environment from the variations of human annotated instructions. For this purpose we resort to pretraining the E.T. language encoder with the synthetic instructions. In particular, we pretrain a seq2seq model to map human instructions into synthetic instructions as described in Section 5.3.3, and study whether it is more beneficial to transfer explicitly the "translated" text or implicitly as representations encoded by the model weights. Our pretraining is done on the original train folds with no additionally generated trajectories. The seq2seq translation performance is very competitive, reaching 97.1% in terms of F1 score. To transfer explicitly the translated (synthetic) instructions, we first train an E.T. agent to follow synthetic instructions from the training folds, and then evaluate the agent on following human instructions by translating these instructions into synthetic ones with our pretrained seq2seq model.

Table 5.7 compares these two pretraining strategies. We can see that both strategies outperform the no pretraining baseline (first row) significantly, and that transferring the encoder works better than explicit translation. For completeness we also report results with BERT pretraining [Devlin et al., 2019] (second row). The BERT model is pretrained on generic text data (e.g.Wikipedia). We use the BERT base model whose weights are released by the authors. We extract its output contextualized word embeddings and use them as the input word embeddings to the language encoder. To our surprise, when compared with the no pretraining baseline, the BERT pretraining decreases the performance in seen environments by 4.4% and brings a marginal improvement of 6.2% relative in unseen environments.

Pretraining	Joint training	Seen	Unseen
		33.8	3.2
✓		37.6	3.8
	✓	38.5	5.4
✓	✓	**46.6**	**7.3**

Table 5.8 – Ablation study of **joint training and language encoder pretraining** with synthetic data. We present baseline results without leveraging synthetic data (first row), the independent performance of pretraining (second row) and joint training (third row), and their combined performance (fourth row).

We conjecture that domain-specific language pretraining is important for the ALFRED benchmark. Overall, these experiments show another advantage of the proposed synthetic annotations and highlights the importance of intermediate language representations to better train instruction-following agents.

We finally combine the language encoder pretraining and the joint training objectives and present the results in Table 5.8. We observe that these two strategies are complementary to each other: the overall relative improvements of incorporating synthetic data over the baseline E.T. model are 37.8% and 228.1% in seen and unseen environments, respectively. We conclude that the synthetic data is especially important for generalization to unseen environments.

We present a complete breakdown of performance improvements with respect to the components added to the LSTM-based baseline model proposed by Shridhar *et al.* [Shridhar et al., 2020]. First, we replace ImageNet visual features with features pretrained to detect objects in ALFRED as explained in Section 5.4.1. Next, we replace explicit pixel mask predictions with a pretrained MaskRCNN model proposed by Singh *et al.* [Singh et al., 2020]. These two components combined bring a significant improvement over the original baseline performance [Shridhar et al., 2020]. We then replace the LSTM model with the E.T. architecture, pretrain the language encoder of the agent to translate human language to synthetic representations, and jointly train the agent using additional 45K demonstrations to achieve the performance reported in Table 5.8.

Components	Seen	Unseen
LSTM baseline (Shridhar *et al.* [Shridhar et al., 2020])	4.8	0.2
+ ALFRED detection pretraining	8.5	0.4
+ Pretrained MaskRCNN [Singh et al., 2020]	23.2	2.4
- LSTM; + Transformer (E.T.)	33.8	3.2
+ Synthetic language pretraining	37.6	3.8
+ Joint training with 45K demonstrations	**46.6**	**7.3**

Table 5.9 – Complete breakdown of performance improvements. We report the performance of the model proposed by Shridhar *et al.* [Shridhar et al., 2020] and sequentially add components that improve its success rate one by one. The components include (1) visual features pretrained to detect objects in ALFRED, (2) a pretrained MaskRCNN to predict pixel masks, (3) the E.T. model, (4) language encoder pretraining on human to synthetic translation, (5) joint training with additional data.

Model	**Validation**		**Test**	
	Seen	Unseen	Seen	Unseen
Shridhar *et al.* [Shridhar et al., 2020]	3.70	0.00	3.98	0.39
Nguyen *et al.* [Van-Quang Nguyen, 2020]	N/A	N/A	12.39	4.45
Singh *et al.* [Singh et al., 2020]	19.15	3.78	22.05	5.30
E.T.	33.78	3.17	28.77	5.04
E.T. (pretr.)	37.63	3.76	33.46	5.56
E.T. (pretr. & joint tr.)	**46.59**	**7.32**	**38.42**	**8.57**
Human performance	-	-	-	91.00

Table 5.10 – Comparison with the models submitted to the public leaderboard on validation and test folds. The highest value per fold is shown in **bold**. 'N/A' denotes that the scores are not reported on the leaderboard or in an associated publication. Our method sets a new state-of-the-art on all metrics.

5.4.4 Comparison with state-of-the-art

We compare the E.T. agent with models from the public leaderboard [*], which have an associated publication. The results on validation and test

*. https://leaderboard.allenai.org/alfred, the results were submitted on February 22, 2021.

Figure 5.4 – A visualization of normalized attention heatmap to previous visual observations, from white (no attention) to red (high attention). In this example, a microwave is first observed at the 8$^{\text{th}}$ timestep, and is highlighted by the visual attention at the 19$^{\text{th}}$ timestep when the agent is asked to put the apple in the microwave.

folds are shown in Table 5.10. The E.T. model trained without synthetic data pretraining and joint training sets a new state-of-the-art on seen environments (row 4). By leveraging synthetic instructions for pretraining, our method outperforms the previous methods [Shridhar et al., 2020, Van-Quang Nguyen, 2020, Singh et al., 2020] and sets a new state-of-the-art on all metrics (row 5). Given additional 45K trajectories for joint training, the E.T. model further improves the results (row 6).

5.4.5 Qualitative results

In this section we provide additional analysis, including visualizations of transformer attention maps with respect to visual and language inputs as well as a few examples of solved and failed tasks.

Visualizing visual attention

To better understand the impact of using a transformer encoder for action predictions, we show several qualitative examples of attention weights produced by the multimodal encoder of an E.T. agent. We use attention rollout [Abnar and Zuidema, 2020] to compute attention weights from an output action to previous visual observations. Attention rollout averages

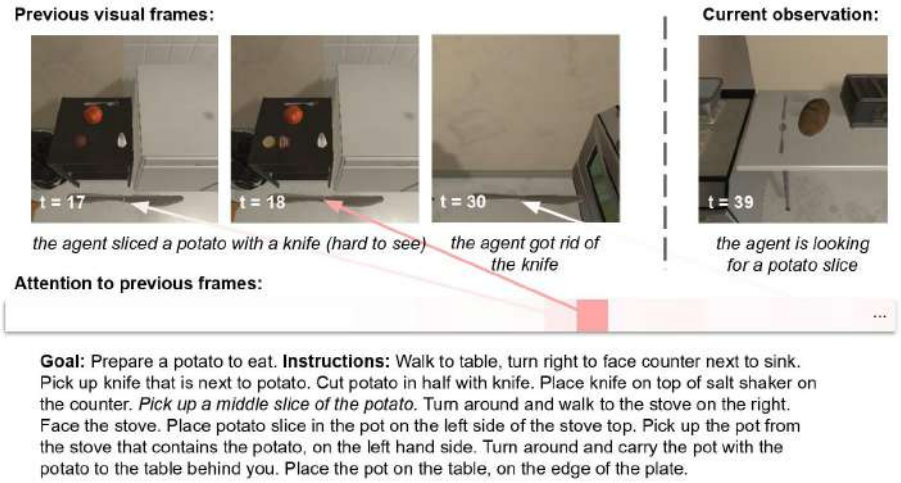


Figure 5.5 – A visualization of normalized attention heatmap to previous visual observations. In this example, the agent is asked to cut a potato (timesteps $17-18$) and to put a slice of it in a pot. At timestep 39 when the agent is asked to retrieve the sliced potato, it attends to frames at timesteps $17-18$ to decide where to go.

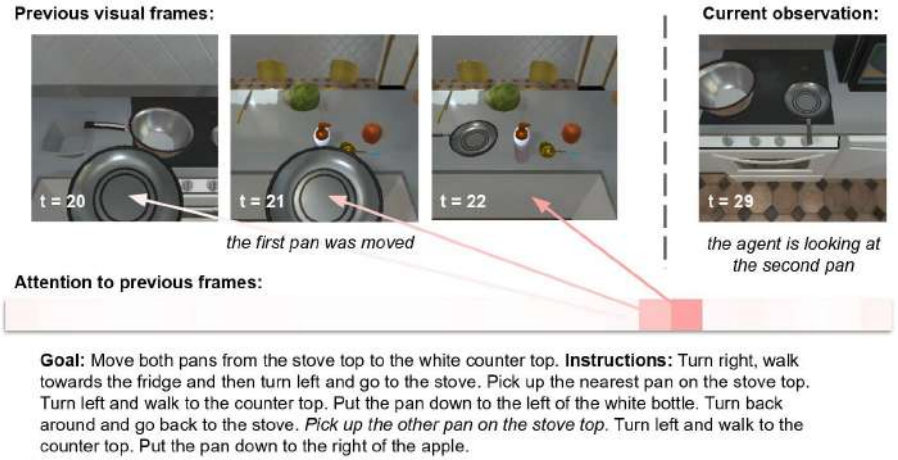


Figure 5.6 – A visualization of normalized attention heatmap to previous visual observations. In this example, the agent is asked to move two identical pans. It moves the first pan at timesteps $20-22$ and attends the frame at timestep 29 when moving the second pan (see the two corresponding pink squares).

Figure 5.7 – A visualization of normalized attention heatmap to previous visual observations. In this example, the agent is asked to wash a cloth and to put it in a cupboard. The agent washes the cloth at timestep 20 but the washed cloth does not look very different from a dirty one. At timestep 31, the agent attends to the previous frames where the washing action is visible to keep track of the cloth state change.

attention of all heads and recursively multiplies attention weights of all transformer layers taking into account skip connections. Figures 5.4-5.7 show examples where an E.T. model attends to previous visual frames to successfully solve a task. The frames attention weights are showed with a horizontal bar where frames corresponding to white squares have close to zero attention scores and frames corresponding to red squares have high attention scores. We do not include the attention score of the current frame as it is always significantly higher than scores for previous frames.

In Figure 5.4 the agent is asked to pick up an apple and to heat it using a microwave. The agent walks past a microwave at timestep 8, picks up an apple at timestep 18 and attends to the microwave frame in order to recall where to bring the apple. In Figure 5.5 the agent slices a potato at timesteps $17-18$ (hard to see on the visual observations). Later, the agent gets rid of the knife and follows the next instruction asking to pick up a potato slice. At timestep 39, the agent attends to the frames $17-18$ where the potato was sliced in order to come back to the slices and complete the task. In Figure 5.6 the agent needs to sequentially move two pans. While picking up the second pan at timestep 29, the agent attends to the frames $20-22$ where the first pan was replaced. In Figure 5.7 the agent is asked to wash a cloth and to put it to a drawer. The agent washes the cloth

at timestep 20 but the cloth state change is hard to notice at the given frames. At timestep 31, the agent attends to the frame with an open tap in order to keep track of the cloth state change. To sum up, the qualitative analysis of the attention mechanism over previous visual frames shows that they are used by the agent to solve challenging tasks and aligns with the quantitative results presented in Section 5.4.3.

Visualizing language attention

Figure 5.8 illustrates transformer attention scores from an output action to input language tokens by comparing two models: (1) E.T. model trained from scratch, (2) E.T. model whose language encoder is pretrained as in Section 5.3.3. Similarly to the visual attention, we use attention rollout and highlight the words with high attention scores with red background color.

In the first example of Figure 5.8, the agent needs to pick up a bat. While the non-pretrained E.T. model has approximately equal attention scores for multiple tokens (those words are highlighted with pale pink color) and does not solve the task, the pretrained E.T. attends to "bat" tokens (highlighted with red) and successfully finds the bat. In the second example, the agent needs to first cool an egg in a fridge and to heat it in a microwave later. The non-pretrained E.T. has the similar attention scores for "microwave" and "refridgerator" tokens (they are highlighted with pink) and makes a mistake by choosing to heat the egg first. The pretrained E.T. agent has higher attention scores for the "refridgerator" tokens and correctly decides to cool the egg first. In the third example, the agent needs to pick up a knife to cut a potato later. The non-pretrained agent distributes its attention over many language tokens and picks up a fork which is incorrect. The pretrained E.T. agent strongly attends to the "knife" token and picks the knife up. The demonstrated examples show that the language pretraining of E.T. results in language attention that is better aligned with human interpretation.

Qualitative analysis

We show 3 successful and 2 failed examples of the E.T. agent solving tasks from the ALFRED validation fold. In Figure 5.9(top) the agent successfully heats an apple and puts it on a table. The agent understands the instruction *"bring the heated apple back to the table on the side"* and navigates back to its previous position. In Figure 5.9(bottom) the agent brings a washed plate to a fridge. The agent does not know where the plate is and

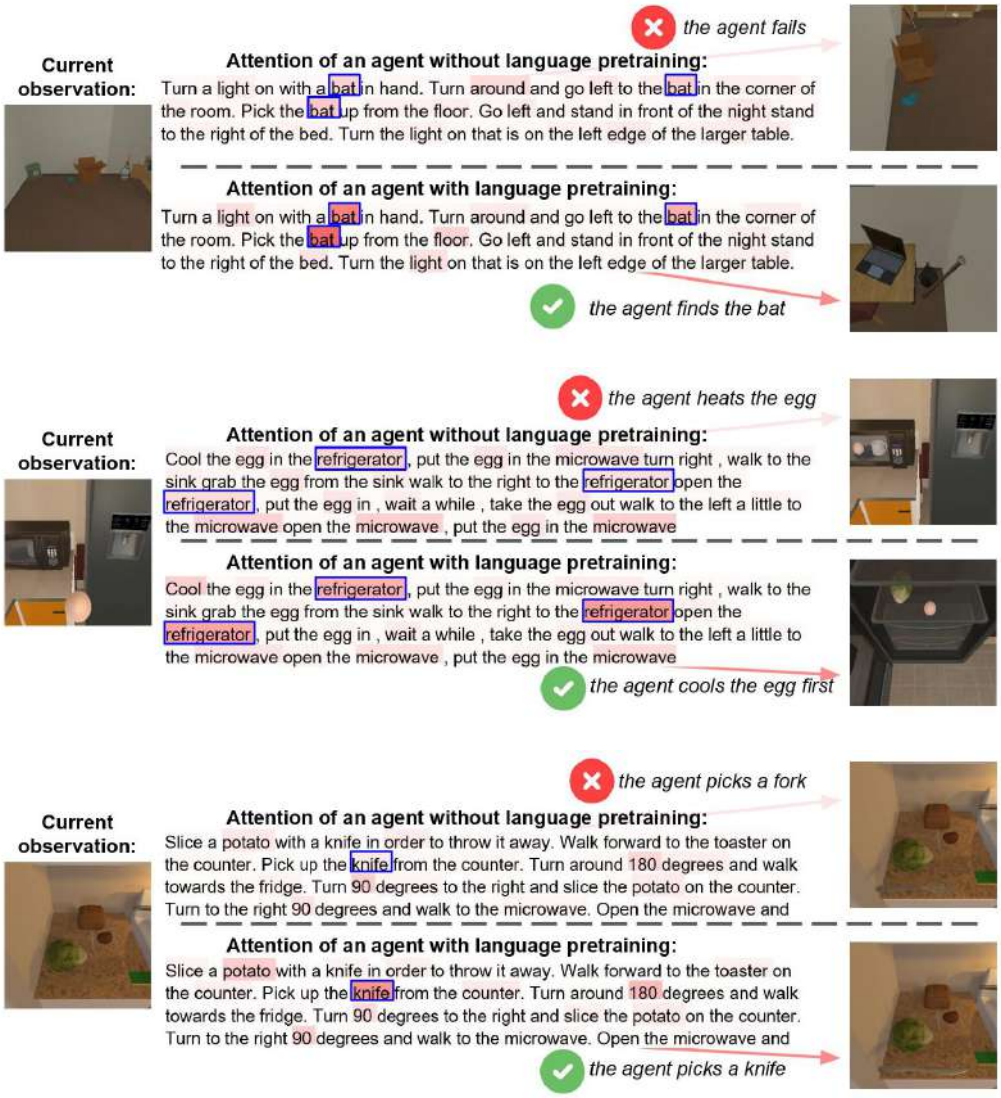
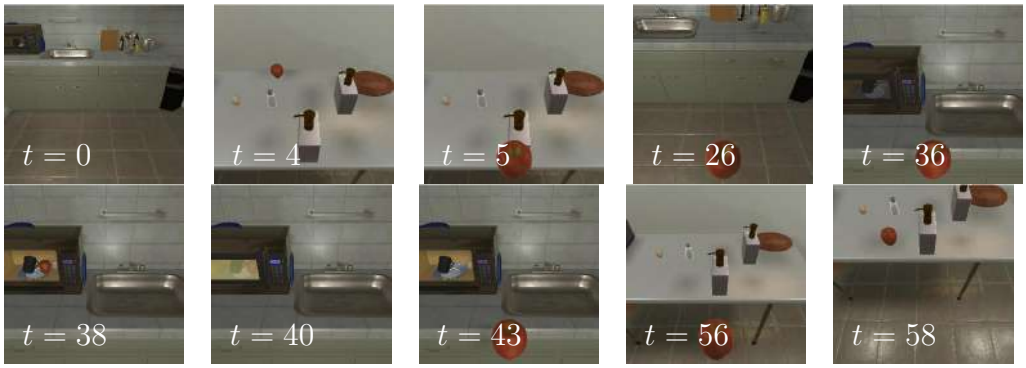
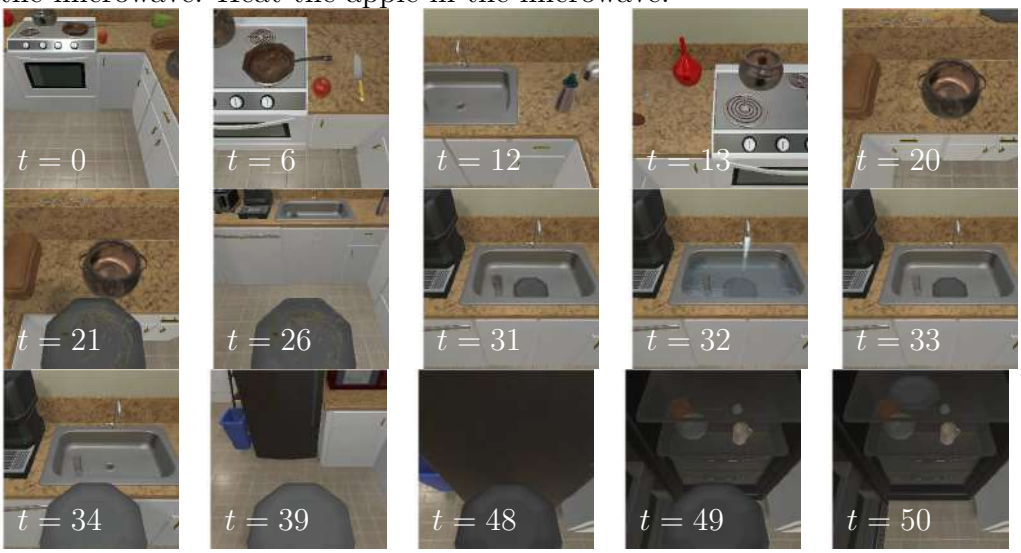


Figure 5.8 – Visualizations of normalized language attention heatmaps, without and with the language encoder pretraining. Red indicates a higher attention score. We observe that the agent trained without language pretraining misses word tokens that are important for the task according to human interpretation (marked with blue rectangles). In contrast, the pretrained E.T. agent often is able to pay attention to those tokens and solve the tasks successfully.

walks along a counter checking several places. Finally, it finds the plate, washes it and brings it to the fridge. In Figure 5.10 the agent performs a

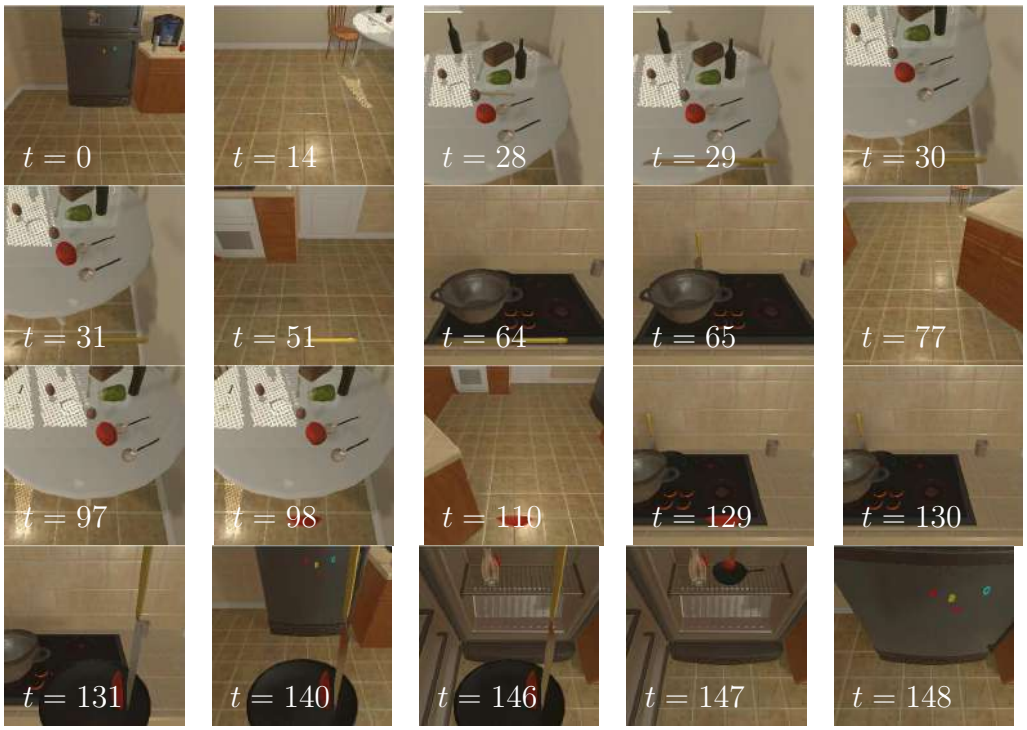


Goal: Put a heated apple on the table. **Instructions**: Turn left and go to the table. Pick up the apple on the table. Go right and bring the apple to the microwave. Heat the apple in the microwave.



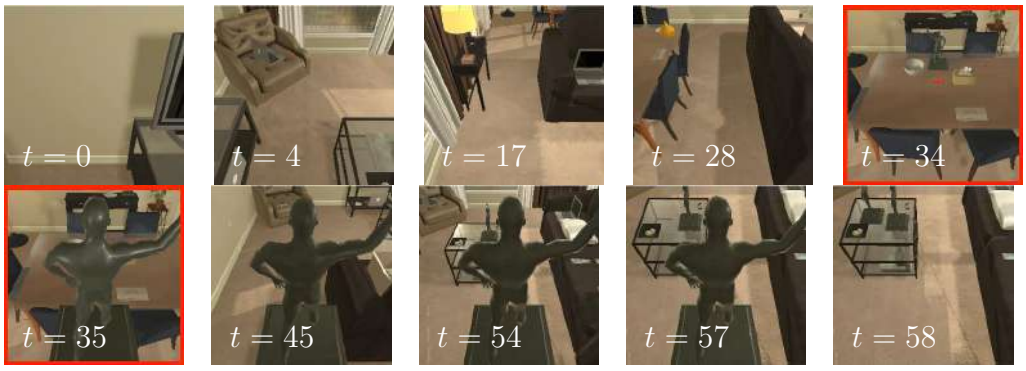
Goal: Place a rinsed plate in the fridge. **Instructions**: Walk ahead to the door, then turn left and take a step, then turn left and face the counter. Pick up the dirty plate on the counter. Walk left around the counter, and straight to the sink. Clean the plate in the sink. Turn left and walk to the fridge. Place the plate on the top shelf of the fridge. Place a pan containing slicing tomato in the refrigerator. *Bring the heated apple back to the table on the side.* Put the heated apple on the table in front of the salt.

Figure 5.9 – Examples of successfully solved tasks. Top: The agent picks up an apple, puts it into a microwave, closes it, turns it on, opens it, picks up the apple again, then navigates *back to the table on the side* and puts the apple on the same table. Bottom: The agent does not know where the dirty plate is and looks at several places on the counter (the first row). It then sees the plate in the corner of the top right image, picks it up, goes to a sink, opens a tap, picks the plate again, navigates to a fridge, opens it and puts the plate to the top shelf of the fridge.

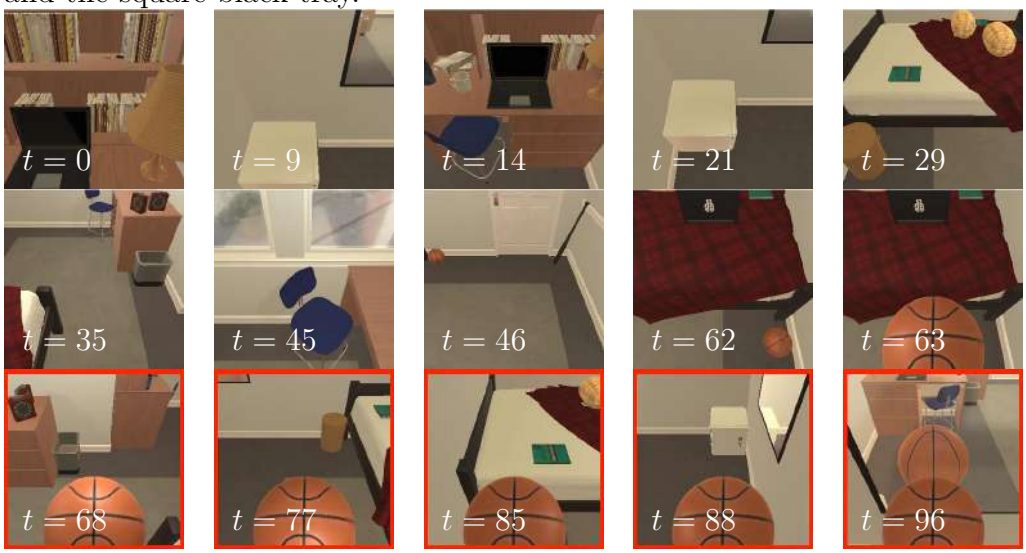


Goal: Place a pan containing slicing tomato in the refrigerator. **Instructions**: Turn right, move to the table opposite the chair. Pick up the knife that is near the tomato. Turn left, move to the table opposite the chair. Slice the tomato that is on the table. Turn left, move to the counter that is left of the bread, right of the potato. Put the knife in the pan. Turn left, move to the table opposite the chair. Pick up a slice of tomato from the counter. Turn left, move to the counter that is left of the bread, right of the potato. Put the tomato slice in the pan. Pick up the pan from the counter. Turn left, move to in front of the refrigerator. Put the pan in the refrigerator.

Figure 5.10 – Example of a successfully solved task. The agent uses 148 actions to complete the task. The agent picks up a knife from a table, slices a tomato in the first image of the second row, brings the knife to a stove, puts the knife on a plate, walks back to the table, grabs a tomato slice, returns to the stove, puts the tomato slice on the same plate, picks up the plate, navigates to a fridge, opens it, puts the plate with the knife and the tomato slice on a shelf and closes the fridge.



Goal: Move a bowl from the table to the coffee table. **Instructions**: Move across the room to the dining room table where the statue is. *Pick up the bowl to the right of the statue on the table.* Carry the bowl to the glass coffee table. Place the bowl on top of the coffee table between the statue and the square black tray.



Goal: Look at a basketball in the lamp light. **Instructions**: Turn around and go to the foot of the bed. Pick up the basketball from the floor. *Turn around and go to the desk in the corner.* Turn on the lamp.

Figure 5.11 – Failure examples. Top: The agent correctly finds both dining and coffee tables but gets confused with *"the bowl to the right of the statue"* reference. The agent decides to pick up a statue instead of a bowl and fails to solve the task. Bottom: The agent is exposed to an unknown environment and fails to follow the navigation instructions. It wanders around the room, eventually finds a basketball but fails to locate a lamp and decides to terminate the episode in front of a mirror.

sequence of 148 actions and successfully solves a task. This example shows that the agent is able to pick up small objects such as a knife and a tomato slice. The agent puts both of them to a plate and brings the plate to a fridge.

Among the most common failure cases are picking up wrong objects and mistakes during navigation. In Figure 5.11(top) the agent misunderstands the instruction *"pick up the bowl to the right of the statue on the table"* and decides to pick up a statue on the frame marked with red. It then brings the statue to a correct location but the full task is considered to be failed. Figure 5.11(bottom) shows a failure mode in an unseen environment. The agent is asked to pick up a basketball and to bring it to a lamp. The agent first wanders around a room but eventually picks up the basketball. It then fails to locate the lamp and finds itself staring into a mirror. The agent gives up on solving the task and decides to terminate the episode.

5.5 Conclusion

In this chapter, we propose Episodic Transformer (E.T.), an architecture for vision-and-language navigation tasks based on the self-attention mechanism. In contrast to commonly used recurrent architectures that rely on a hidden state, E.T. observes the complete history of vision, language, and action inputs and encodes it with a multimodal transformer. We quantitatively and qualitatively demonstrate that providing the whole history of observations and actions is important for agent performance on compositional tasks. On the challenging ALFRED benchmark, E.T. outperforms competitive recurrent baselines and achieves state-of-the-art performance. We also propose to use synthetic instructions as intermediate representations for language-only pretraining and joint training with human annotated instructions. Given the synthetic instructions, the performance of E.T. is further improved in seen and especially, in unseen environments.

6

Conclusion

Our work proposes novel approaches for learning control policies for real robots. Our experiments demonstrate that control for real-world robots can be learned using only synthetic images and a limited amount of domain knowledge. We also made a step towards bridging the gap between robot learning and natural language processing by proposing a new approach to solve a language-conditioned navigation task.

In Section 6.1, we summarize key contributions made in this dissertation and main observations of our experiments. In Section 6.2, we discuss research directions which can be considered for future work.

6.1 Summary of contributions

Learning to augment synthetic images for sim2real policy transfer. Our first contribution is a method to learn augmentation functions for simulation-to-reality policy transfer. We study how a proxy task of object position estimation and a small amount of real images can be used to find a simulation-to-reality function. This function consists of a sequence of random transformations such as scaling and adding noise and is used to augment synthetic images. Once augmented, synthetic images and corresponding expert actions can be used for policy learning with the behavior cloning approach. The control policies learned on augmented synthetic images can be used directly on a real robot. Our experiments suggest that the found simulation-to-reality augmentation function is task-independent and can be used to learn control policies for a range of manipulation tasks. By training policies on a distribution of objects and their shapes in simulated scenes, our approach allows real-world policies to be robust to unseen during training objects. Our experiments also indicate that the performance

on the proxy task strongly correlates with the real-world performance of the learned policies. Notably, our method only requires real images for the proxy task of object position estimation and does not require any real images of downstream manipulation tasks. We evaluate our method on three real-world manipulation tasks including picking, stacking and placing. We further apply our method on more complex tasks such as a task of pouring of several objects and an assembly task.

Learning to combine primitive skills.
Our second contribution is a method to learn combinations of primitive skills. We propose to pretrain skills with behavior cloning and show how to significantly decrease the number of required demonstrations by exploring recent CNN architectures and data augmentation. We then propose to learn a master policy that switches between skills using reinforcement learning and a task completion signal. In contrast to prior approaches, our method requires neither full-task demonstrations nor intermediate rewards. We demonstrate that the skill policies learned with behavior cloning can immediately react to changes in the scene and the master policy learned through trial-and-error can replan in case of a skill failure. Our approach also employs the previously proposed method for sim2real transfer and works on a real robot while using only synthetic images for training. We show excellent results on simulated tasks and their real-world counterparts. We also demonstrate the versatility of our approach in challenging real-world settings with dynamic scene changes which include changes of object positions, temporary object occlusions, and unseen during training object instances.

Learning visual policies for building 3D shape categories.
Our third contribution is a method to assembly 3D object shapes using a robotic arm and varying sets of building blocks. To solve this challenging task using only a final configuration example for supervision, we propose a two-phase approach. First, our approach solves the task in a low-dimensional state space of a simulator by disassembly. We suggest that disassembly trajectories are easier to sample and they can be treated as assembly demonstrations once reversed. We also propose to transfer low-dimensional state-space demonstrations to the visual observation space of a robot by using a renderer of the simulator. In the second phase of the method, we train visual policies using behavior cloning and automatically collected assembly demonstrations. Similar to our previous approaches, we apply the sim2real augmentation function and use only synthetic images to train real-world policies. We demonstrate the reactive ability of our method to re-assemble objects using additional primitives introduced to the real seen

and the robust performance of our policy for unseen primitives resembling building blocks used during training. Our visual assembly policies reach up to 95% success rate when evaluated on a real robot.

Episodic Transformer for vision-and-language navigation.
Our fourth contribution is a method for solving vision-and-language navigation tasks called Episodic Transformer (E.T.). We propose a novel transformer-based architecture which is able to solve compositional sequences of subtasks. At any time, the poposed architecture observes a complete history of language instructions, vision observations, and previous actions and encodes them with a multimodal transformer. We quantitatively and qualitatively show that providing the whole history of observations and actions is important for agent performance. On the challenging ALFRED benchmark, our approach outperforms competitive recurrent baselines and achieves state-of-the-art performance. We also propose to use synthetic instructions as intermediate representations to decouple understanding the visual appearance of an environment from the variations of natural language instructions. The synthetic instructions are used for language-only pretraining and joint training together with human-annotated instructions. We experimentally show that the synthetic instructions improve performance on seen and especially unseen environments and help our model to improve state-of-the-art results on both of them.

6.2 Perspectives for future research

In this section, we introduce several ideas about future extensions of our work, which are suggested by our experiments and recent developments in the fields of robot learning, computer vision, and machine learning.

6.2.1 Learning to augment synthetic images for sim2real policy transfer

Extending sim2real transfer to RGB data. The proposed approach finds a sequence of random transformations for depth images to transfer learned control policies from simulation to reality. Previous work found that designing such transformations manually could lead to a successful policy transfer for RGB images [Tobin et al., 2017]. We hypothesize that applying our method of Monte Carlo tree search over possible sequences of transformations for RGB data could result in optimal sim2real augmentations and potentially outperform the mentioned manually designed transfor-

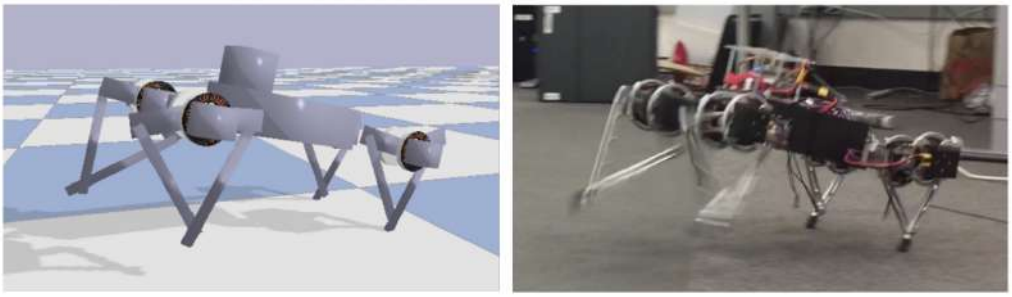


Figure 6.1 – Our sim2real approach does not consider the differences between simulation and reality in terms of physical behavior of synthetic objects. Such differences may be critical for a successful transfer of policies in contact-rich tasks. The figure shows an example of such task where the left part shows a simulated legged robot and the right part shows its real-world correspondance [Tan et al., 2018].

mations. Such transformations for RGB data could include random color operations with already rendered images such as contrast and brightness change [Cubuk et al., 2019] and explicitely rendered appearance changes such as texture swaps. Once our method is extended to the RGB data, it could be further extended for RGB-D images by applying the transformations to corresponding image channels.

Addressing the physical sim2real gap. While our work focuses on visual differences between simulated and real images, we do not consider possible differences of simulation from the real world in terms of the physical behavior of synthetic objects. Although we did not observe the impact of such differences for our manipulation tasks, the physical sim2real gap might play a key role in transferring policies for contact-rich manipulations and locomotion [Liu et al., 2019] (see Figure 6.1). Apart from designing high-fidelity simulations of controllers and sensors using system identification methods, previous work proposed to use domain randomization approaches and to manually design sensor noise and perturbations in synthetic environments [Tan et al., 2018]. One can also consider extending our approach by adding transformations that impact physical behavior of simulated objects and robots such as friction coefficients changes and mass perturbations. In this case, a major challenge may lie in reducing the cost of the data generation to make the sim2real function search feasible using limited resources.

Making the method to be robot platform agnostic. The proposed method can handle several differences between simulation and reality including a camera position, background clutter, and novel objects

unseen during training. However, we assume that a specific robotic arm is used which is simulated using an exact model provided by its manufacturer. Previous work showed that it is possible to train a control policy that works on a range of robotics platforms using a small amount of data for adapation [Nagabandi et al., 2019, Ghadirzadeh et al., 2021]. In theory, it should be possible to leverage a simulator to learn a single control policy for several robotics arms which, for instance, have the same number of degrees of freedom. One could use our method and design a set of random transformations for the arm configuration, its visual appearance and physical behavior. To facilitate the learning process, control policies might be conditioned on the current arm configuration and the robot action space can be parameterized independently of the manipulator's internal state using the end-effector position.

6.2.2 Learning to combine primitive skills

Learning multiple tasks with shared skills. The proposed approach assumes learning a master policy to combine a set of skills. The master policy is learned through trial-and-error and skill policies are pretrained from short demonstrations. Although our method can autonomously decide which skills to use and which to ignore for a given task, the provided demonstrations of skills are specifically designed using knowledge of which subtasks the current task can be decomposed into. Moreover, some tasks may have an intersecting vocabulary of skills required for a successful solution [Farahani and Mozayani, 2020]. A possible extension of our method would be using it for solving multiple tasks given a broad set of skills. Such extension would introduce a more challenging problem from the RL exploration point of view and would result in longer training time. This difficulty could be addressed using exploration heuristics [Tang et al., 2017, Pathak et al., 2017] or more sample-efficient RL approaches including off-policy and model-based methods [Fujimoto et al., 2019, Hafner et al., 2019].

Learning soft combinations of skills. While our proposed approach assumes hard skill switching by the master policy, i.e. only one skill policy can be chosen by the master at any given time, the approach could benefit from being able to combine several skills. Our preliminary work demonstrated that in some cases choosing one skill at a time (shown on the left in Figure 6.2) may results in suboptimal behavior and can be outperformed by predicting a modulation signal with the master policy instead [Pashevich et al., 2018]. One of the examples of such modulation signal is the coefficients to weigh actions of individual skills with (shown on the right in Figure 6.2). At the same time, using several skills at once can allow

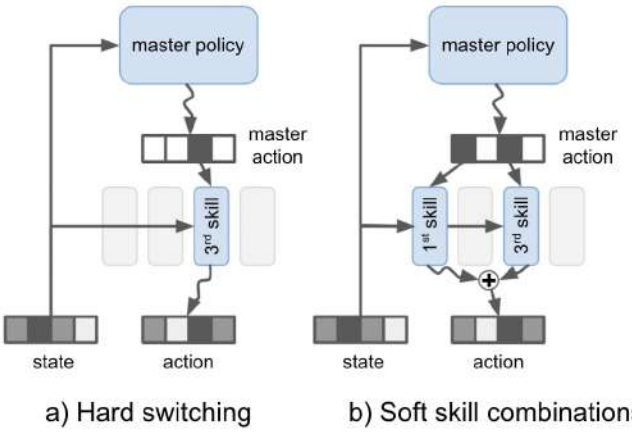


Figure 6.2 – Our preliminary work demonstrated that soft combinations of skills (right) can outperform hard skill switching (left) which can be applied in the context of our proposed method [Pashevich et al., 2018].

for a further increase in versatility by employing lower-level skills such as movements in orthogonal directions [Florensa et al., 2017a]. This future work follows the same direction as the previous proposal and could further increase the generalizability of our approach.

Unsupervised skill discovery. Our approach assumes that each of the demonstrations used to learn skill policies has a skill label. Unsupervised skill discovery given unlabeled demonstrations or even less structured data such as self-play [Lynch et al., 2019] would allow to futher decrease the amount of domain-specific prior knowledge. In our preliminary work [Pashevich et al., 2018], we found that a hierarchy of RL policies can divide a manipulation task into skill subtasks such as reaching, grasping, and lifting without any additional supervision. Another approach would be to use clustering techniques such as the EM approach [Greff et al., 2017] to divide long demonstrations into short skills and to train skill policies with the cluster labels [Karoly et al., 2019]. A potential difficulty would be to find the optimal granularity of discovered skills, in other words, to decide how high- or low-level they should be.

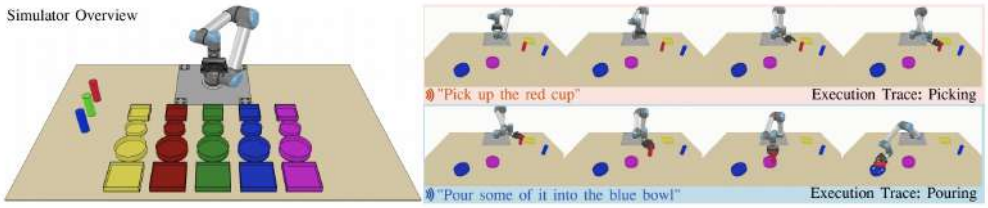
6.2.3 Learning visual policies for building 3D shape categories

Extending the method for physically irreversible actions. The proposed approach disassembles 3D object shapes and reverses applied actions to get assembly demonstrations. While this disassembling procedure

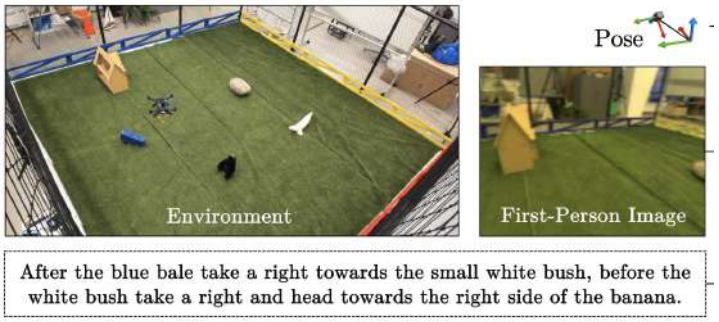
may not be directly applicable to all tasks out of the box, we believe that it could generalize even to physically irreversible actions in tasks such as preparing a meal and more complex assembling tasks involving, for example, drilling. This can be done by learning an appropriate backward model in simulation [Agostinelli et al., 2019, Edwards et al., 2018] and using it for the action reversal. For instance, in the task of making tea or, more precisely, pouring water into a cup, the "unpouring" action is not feasible in the real world. However, one could predict the result of "unpouring" from a cup full of water using a learned backward model. Next, a simulator can be used to generate an observation with an empty cup and water in a kettle. Therefore, it would be possible to generate demonstrations of tea preparation in simulation, use them to learn a control policy and transfer it for a real-world tea preparation task.

Reducing the method complexity. To find the shortest assembly demonstration from a given state, our proposed approach extensively searches for all possible disassembly paths and merges them into a demonstration. The complexity of such procedure grows exponentially with the number of primitives used in the scene. While this did not become a bottleneck for the problems considered in our work, more complex tasks may require significantly longer time to generate demonstrations for. For such cases, one can extend our method to use an approximation of the extensive search by sampling a fixed number of disassembly trajectories. Apart from reducing the wall time required for generating demonstrations, one could increase the sample efficiency of the proposed approach in terms of the number of required demonstrations by using more recent off-policy RL methods [Espeholt et al., 2018, Munos et al., 2016].

Generalizing the category definition. Our work proposes a novel method to learn control policies able to build objects that have a certain 3D shape which we refer to as a category. The trained policy is required to learn what the given 3D shape is. In our work, we consider tasks of building relatively simple 3D shapes of towers and arches. We believe that it is possible to make an agent reason about more sophisticated concepts such as, for example, the concept of a house which means a few connected walls equipped with a door and covered with a roof. Making the agent learn such rules might be simpler than providing it with a large number of simulated examples of various houses. Another interesting example of an extension for the category definition and our method application could be the furniture assembly task where the agent is asked to build, for instance, an object that belongs to a chair category [Suárez-Ruiz et al., 2018].



(a) Language-conditioned manipulation task [Stepputtis et al., 2020].



(b) Instruction-following task with a physical quadcopter [Blukis et al., 2019].

Figure 6.3 – We believe that the E.T. architecture can be applied to solve multimodal robotics problems where the task is defined using natural language instructions.

6.2.4 Episodic Transformer for vision-and-language navigation

Extending the E.T. architecture to arbitrarily long tasks. One of the main advantages of the E.T. architecture is the capacity to encode a full history of previous observations and actions. Despite the benefits shown by our experiments, this feature comes at the cost of lengthy input sequences which might become prohibitively expensive in longer tasks. To this extend, several solutions can be proposed starting from simply using only a fixed number of the latest observations and actions. Moreover, novel attention-based models extending transformers for handling much longer input sequences [Zaheer et al., 2020, Dai et al., 2019] can be used. Our model also inherits the quadratic complexity of the transformer architecture with respect to the input length which might become a bottleneck to use it in real time. The same class of mentioned above transformer architectures could be considered to decrease the complexity to linear [Zaheer et al., 2020, Katharopoulos et al., 2020].

Applying the E.T. architecture in the context of robotics. While our work is not directly applied to robotics, we consider it to be a step

towards bridging the gap between developments in robot learning and natural language processing. The proposed approach is shown to work in the context of vision-and-language navigation and we believe that it is generalizable to other control tasks where an agent receives a multimodal input such as language commands and visual observations. Examples of this scenario include manipulation tasks with a robotic arm defined by natural language [Stepputtis et al., 2020] shown in Figure 6.3a and controlling a quadcopter to reach goals defined by textual instructions [Blukis et al., 2019] shown in Figure 6.3b. The major challenges of applying the E.T. architecture for such tasks lie in a potential change of discrete actions to continuous control and higher requirements for the visual perception caused by the usage of real-world images. In addition, the previously mentioned adaptations for handling long inputs might be necessary depending on the robot control frequency and a task length.

Improving model performance by using other input modalities. The proposed E.T. architecture uses a multimodal encoder on top of separate encoders for each of three input modalities: language, images, and actions. Transformer-based architectures were shown to successfully work with various types of modalities including language, RGB images, videos, speech, hyperspectral images, and others [Sun et al., 2019, He et al., 2020, Gabeur et al., 2020]. Recently, novel datasets were proposed with potential applications of other data sources such as human speech [Ku et al., 2020] to improve the performance of control policies. One could equip the proposed E.T. architecture with another encoder to solve tasks where speech is also available. Another promising extension would be providing the model with a map of an environment which could be treated as one more token of another modality. Such map can be either provided as a ground truth map if it is available [Chen et al., 2021] or a map estimated on the fly [Chaplot et al., 2020, Narasimhan et al., 2020]. The map modality could be especially important to improve the navigation performance and close the gap between seen and unseen environments.

A

Publications

This thesis has led to several publications summarized below.

International Conferences

- A. Pashevich*, R. Strudel*, I. Kalevatykh, I. Laptev, C. Schmid
  Learning to Augment Synthetic Images for Sim2Real Policy Transfer
  Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2019, [Pashevich et al., 2019a].
  https://hal.archives-ouvertes.fr//hal-02273326v1

- R. Strudel*, A. Pashevich*, I. Kalevatykh, I. Laptev, J. Sivic, C. Schmid
  Learning to combine primitive skills: A step towards versatile robotic manipulation
  Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2020, [Strudel et al., 2020b].
  https://hal.archives-ouvertes.fr/hal-02274969v1

- A. Pashevich*, I. Kalevatykh*, I. Laptev, C. Schmid
  Learning visual policies for building 3D shape categories
  Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2020, [Pashevich et al., 2020a].
  https://hal.archives-ouvertes.fr/hal-02945024v1

- A. Pashevich, C. Schmid, C. Sun
  Episodic Transformer for Vision-and-Language Navigation
  Proceedings of the International Conference on Computer Vision

∗. Equal contribution

(ICCV) 2021, [Pashevich et al., 2021a].
https://hal.archives-ouvertes.fr/hal-03371803v1

Other Publications

- A. Pashevich, D. Hafner, J. Davidson, R. Sukthankar, C. Schmid
  Modulated Policy Hierarchies
  Deep Reinforcement Learning Workshop, in conjunction with the Conference on Neural Information Processing Systems (NeurIPS) 2018, [Pashevich et al., 2018].
  https://hal.archives-ouvertes.fr/hal-01963580v1

B

Software

Several code bases created over the course of the PhD are available online.

- Source code of sim2real and RLBC approaches:
  https://github.com/rstrudel/rlbc
  The code allows to train control policies for several manipulation tasks and to transfer them to a real-world UR5 robot. The code implements skill policies pretraining with behavior clonning, master policy training with Proximal Policy Optimization, sim2real augmentation function optimization procedure with Monte Carlo Tree Search, and scalable data collection and policy evaluation tools. The code also allows to implement other manipulation tasks and train control policies for them. The documentation provides exact steps to reproduce the results of two our papers [Pashevich et al., 2019a, Strudel et al., 2020b]. The code is written using PyTorch library.

- Source code of the Episodic Transformer (E.T.) approach:
  https://github.com/alexpashevich/E.T.
  The code allows to train a VLN agent for the ALFRED benchmark. The code implements the transformer-based E.T. architecture and a scalable way to evaluate it on validation and test folds of ALFRED. The repository contains links to pretrained models and additional ALFRED trajectories used for their training. The documentation provides exact steps to reproduce our results [Pashevich et al., 2021a]. The code is written using PyTorch library.

Bibliography

[Abnar and Zuidema, 2020] Abnar, S. and Zuidema, W. (2020). Quantifying attention flow in transformers. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[Agostinelli et al., 2019] Agostinelli, F., McAleer, S., Shmakov, A., and Baldi, P. (2019). Solving the rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*.

[Agrawal et al., 2016] Agrawal, P., Nair, A. V., Abbeel, P., Malik, J., and Levine, S. (2016). Learning to poke by poking: Experiential learning of intuitive physics. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Allen, 1995] Allen, J. (1995). *Natural Language Understanding*. Benjamin / Cummings Publishing Company.

[Anderson et al., 2018] Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and van den Hengel, A. (2018). Vision-and-Language Navigation: Interpreting visually-grounded navigation instructions in real environments. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Andreas and Klein, 2015] Andreas, J. and Klein, D. (2015). Alignment-based compositional semantics for instruction following. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Andrychowicz et al., 2017] Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R. H., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *Advances in Neural Information Processing Systems (NeurIPS)*.

[Argall et al., 2009] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *IEEE Robotics and Automation Society (RAS).*

[Artzi and Zettlemoyer, 2013] Artzi, Y. and Zettlemoyer, L. (2013). Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL).*

[Atkeson and Schaal, 1997] Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *International Conference on Machine Learning (ICML).*

[Bacon et al., 2017] Bacon, P.-L., Harb, J., and Precup, D. (2017). The Option-Critic Architecture. In *Conference on Artificial Intelligence (AAAI).*

[Belinkov and Glass, 2019] Belinkov, Y. and Glass, J. R. (2019). Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics (TACL).*

[Bellemare et al., 2016] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS).*

[Berant et al., 2013] Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic parsing on freebase from question-answer pairs. In *Conference on Empirical Methods in Natural Language Processing (EMNLP).*

[Berner et al., 2019] Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J. W., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint.*

[Blukis et al., 2019] Blukis, V., Terme, Y., Niklasson, E., Knepper, R. A., and Artzi, Y. (2019). Learning to map natural language instructions to physical quadcopter control using simulated flight. In *Conference on Robot Learning (CoRL).*

[Boden, 2008] Boden, M. (2008). Mind As Machine: A History of Cognitive Science. *Oxford University Press.*

[Bojarski et al., 2017] Bojarski, M., Yeres, P., Choromańska, A., Choromanski, K., Firner, B., Jackel, L., and Muller, U. (2017). Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv preprint*.

[Bollini et al., 2013] Bollini, M., Tellex, S., Thompson, T., Roy, N., and Rus, D. (2013). Interpreting and executing recipes with a cooking robot. *Experimental Robotics*.

[Bousmalis et al., 2018] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., Levine, S., and Vanhoucke, V. (2018). Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Branavan et al., 2009] Branavan, S. R., Chen, H., Zettlemoyer, L. S., and Barzilay, R. (2009). Reinforcement learning for mapping instructions to actions. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[Brown et al., 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Bugmann et al., 2001] Bugmann, G., Lauria, S., Kyriacou, T., Klein, E., Bos, J., and Coventry, K. (2001). Using verbal instructions for route learning: Instruction analysis. In *Towards Intelligent Mobile Robots (TIMR)*.

[Burda et al., 2019] Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2019). Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations (ICLR)*.

[Calandra et al., 2015] Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. (2015). Bayesian optimization for learning gaits under uncertainty. In *Annals of Mathematics and Artificial Intelligence*.

[Carion et al., 2020] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European Conference on Computer Vision (ECCV)*.

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint*.

[Chaplot et al., 2020] Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R. (2020). Learning to Explore using Active Neural SLAM. In *International Conference on Learning Representations (ICLR)*.

[Chen and Mooney, 2011] Chen, D. and Mooney, R. (2011). Learning to interpret natural language navigation instructions from observations. In *Conference on Artificial Intelligence (AAAI)*.

[Chen et al., 2019a] Chen, H., Suhr, A., Misra, D., Snavely, N., and Artzi, Y. (2019a). Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Chen et al., 2021] Chen, K., Chen, J. K., Chuang, J., V'azquez, M., and Savarese, S. (2021). Topological planning with transformers for vision-and-language navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Chen et al., 2019b] Chen, Z., Guo, D., Xiao, T., Xie, S., Chen, X., Yu, H., Gray, J., Srinet, K., Fan, H., Ma, J., Qi, C., Tulsiani, S., Szlam, A., and Zitnick, L. (2019b). Order-Aware Generative Modeling Using the 3D-Craft Dataset. In *International Conference on Computer Vision (ICCV)*.

[Chen and Huang, 2017] Chen, Z. and Huang, X. (2017). End-to-end learning for lane keeping of self-driving cars. In *IEEE Intelligent Vehicles Symposium (IV)*.

[Cheng et al., 2018] Cheng, C.-A., Yan, X., Wagener, N., and Boots, B. (2018). Fast policy learning through imitation and reinforcement. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

[Clark, 2015] Clark, A. (2015). Python Imaging Library (PIL). https://pillow.readthedocs.io/en/stable/.

[Clark and Amodei, 2016] Clark, J. and Amodei, D. (2016). Faulty reward functions in the wild. https://openai.com/blog/faulty-reward-functions/.

[Codevilla et al., 2018] Codevilla, F., Miiller, M., López, A., Koltun, V., and Dosovitskiy, A. (2018). End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Côté et al., 2018] Côté, M.-A., Kádár, Á., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Hausknecht, M., El Asri, L., Adada, M., et al. (2018). Textworld: A learning environment for text-based games. In *Workshop on Computer Games*.

[Coulom, 2006] Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. *Computers and Games*.

[Coumans, 2009] Coumans, E. (2009). Bullet physics engine. https://bulletphysics.org.

[Cubuk et al., 2019] Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. (2019). AutoAugment: Learning Augmentation Policies from Data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Dahl et al., 2012] Dahl, G. E., Yu, D., Deng, L., and Acero, A. (2012). Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition. In *IEEE Transactions on Audio, Speech, and Language Processing*.

[Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

[Dantam et al., 2018] Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research (IJRR)*.

[Das et al., 2018] Das, A., Gkioxari, G., Lee, S., Parikh, D., and Batra, D. (2018). Neural modular control for embodied question answering. In *Conference on Robot Learning (CoRL)*.

[Dayan and Hinton, 1993] Dayan, P. and Hinton, G. (1993). Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[de Haan et al., 2019] de Haan, P., Jayaraman, D., and Levine, S. (2019). Causal confusion in imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Deitke et al., 2020] Deitke, M., Han, W., Herrasti, A., Kembhavi, A., Kolve, E., Mottaghi, R., Salvador, J., Schwenk, D., VanderBilt, E., Wallingford, M., Weihs, L., Yatskar, M., and Farhadi, A. (2020). Robothor: An open simulation-to-real embodied ai platform. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

[DeVries and Taylor, 2017] DeVries, T. and Taylor, G. W. (2017). Improved Regularization of Convolutional Neural Networks with Cutout. *arXiv preprint*.

[Ding et al., 2020] Ding, D., Hill, F., Santoro, A., and Botvinick, M. (2020). Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures. *arXiv preprint*.

[Dosovitskiy et al., 2021] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*.

[Duan et al., 2017] Duan, Y., Andrychowicz, M., Stadie, B., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Duan et al., 2016] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv preprint*.

[Dvornik et al., 2018] Dvornik, N., Mairal, J., and Schmid, C. (2018). Modeling Visual Context is Key to Augmenting Object Detection Datasets. In *European Conference on Computer Vision (ECCV)*.

[Ebert et al., 2018] Ebert, F., Finn, C., Dasari, S., Xie, A., Lee, A. X., and Levine, S. (2018). Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control. *arXiv preprint*.

[Edelkamp and Hoffmann, 2004] Edelkamp, S. and Hoffmann, J. (2004). PDDL2.2: The Language for the Classical Part of IPC-4. *Technical Report, University of Freiburg*.

[Edwards et al., 2018] Edwards, A., Downs, L., and Davidson, J. C. (2018). Forward-backward reinforcement learning. In *ICRA Machine Learning in Planning and Control of Robot Motion Workshop*.

[Eitel et al., 2015] Eitel, A., Springenberg, J. T., Spinello, L., Riedmiller, M. A., and Burgard, W. (2015). Multimodal deep learning for robust RGB-D object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Erez et al., 2015] Erez, T., Tassa, Y., and Todorov, E. (2015). Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Espeholt et al., 2018] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning (ICML)*.

[Espiau et al., 1992] Espiau, B., Chaumette, F., and Rives, P. (1992). A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*.

[Farahani and Mozayani, 2020] Farahani, M. D. and Mozayani, N. (2020). Evaluating skills in hierarchical reinforcement learning. *International Journal of Machine Learning and Cybernetics*.

[Fikes and Nilsson, 1971] Fikes, R. and Nilsson, N. (1971). Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*.

[Finn et al., 2017] Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*.

[Finn et al., 2016] Finn, C., Tan, X., Duan, Y., Darrell, T., Levine, S., and Abbeel, P. (2016). Deep spatial autoencoders for visuomotor learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Florensa et al., 2017a] Florensa, C., Duan, Y., and Abbeel, P. (2017a). Stochastic Neural Networks for Hierarchical Reinforcement Learning. In *International Conference on Learning Representations (ICLR)*.

[Florensa et al., 2017b] Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017b). Reverse curriculum generation for reinforcement learning. In *Conference on Robot Learning (CoRL)*.

[Frans et al., 2018] Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. (2018). Meta Learning Shared Hierarchies. In *International Conference on Learning Representations (ICLR)*.

[Fried et al., 2018] Fried, D., Hu, R., Cirik, V., Rohrbach, A., Andreas, J., Morency, L.-P., Berg-Kirkpatrick, T., Saenko, K., Klein, D., and Darrell, T. (2018). Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Fujimoto et al., 2019] Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning (ICML)*.

[Gabeur et al., 2020] Gabeur, V., Sun, C., Alahari, K., and Schmid, C. (2020). Multi-modal transformer for video retrieval. In *European Conference on Computer Vision (ECCV)*.

[Gandhi et al., 2017] Gandhi, D., Pinto, L., and Gupta, A. (2017). Learning to fly by crashing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Gao et al., 2018] Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., and Darrell, T. (2018). Reinforcement Learning from Imperfect Demonstrations. In *International Conference on Learning Representations (ICLR) workshop*.

[Garrett et al., 2015] Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2015). FFRob: An efficient heuristic for task and motion planning. *Algorithmic Foundations of Robotics XI*.

[Garrett et al., 2018a] Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2018a). FFRob: Leveraging symbolic planning for efficient task and motion planning. *The International Journal of Robotics Research (IJRR)*.

[Garrett et al., 2018b] Garrett, C. R., Lozano-Pérez, T., and Kaelbling, L. P. (2018b). Sampling-based methods for factored task and motion planning. *The International Journal of Robotics Research (IJRR)*.

[Ghadirzadeh et al., 2021] Ghadirzadeh, A., Chen, X., Poklukar, P., Finn, C., Björkman, M., and Kragic, D. (2021). Bayesian meta-learning for few-shot policy adaptation across robotic platforms. In *arXiv preprint*.

[Ghallab et al., 1998] Ghallab, M., Howe, A., Knoblock, C., Mcdermott, D., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL: The Planning Domain Definition Language.

[Girdhar et al., 2019] Girdhar, R., Carreira, J., Doersch, C., and Zisserman, A. (2019). Video action transformer network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Gopalan et al., 2018] Gopalan, N., Arumugam, D., Wong, L., and Tellex, S. (2018). Sequence-to-Sequence Language Grounding of Non-Markovian Task Specifications. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Grabner et al., 2018] Grabner, A., Roth, P. M., and Lepetit, V. (2018). 3D Pose Estimation and 3D Model Retrieval for Objects in the Wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Graves et al., 2016] Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., Colmenarejo, S. G., Grefenstette, E., Ramalho, T., Agapiou, J., Badia, A. P., Hermann, K. M., Zwols, Y., Ostrovski, G., Cain, A., King, H., Summerfield, C., Blunsom, P., Kavukcuoglu, K., and Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*.

[Greff et al., 2017] Greff, K., van Steenkiste, S., and Schmidhuber, J. (2017). Neural expectation maximization. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Gu et al., 2016] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2016). Deep Reinforcement Learning for Robotic Manipulation. In *International Conference on Machine Learning (ICML)*.

[Gupta et al., 2018] Gupta, A., Murali, A., Gandhi, D., and Pinto, L. (2018). Robot Learning in Homes: Improving Generalization and Reducing Dataset Bias. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Ha et al., 2018] Ha, S., Kim, J., and Yamane, K. (2018). Automated Deep Reinforcement Learning Environment for Hardware of a Modular Legged Robot. In *International Conference on Ubiquitous Robots (UR)*.

[Haarnoja et al., 2018] Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. (2018). Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[Haarnoja et al., 2017] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*.

[Hadfield-Menell et al., 2017] Hadfield-Menell, D., Milli, S., Abbeel, P., Russell, S. J., and Dragan, A. (2017). Inverse reward design. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Hafner et al., 2019] Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019). Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning (ICML)*.

[Handa et al., 2016] Handa, A., Patraucean, V., Badrinarayanan, V., Stent, S., and Cipolla, R. (2016). Scenenet: Understanding real world indoor scenes with synthetic data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Hao et al., 2020] Hao, W., Li, C., Li, X., Carin, L., and Gao, J. (2020). Towards learning a generic agent for vision-and-language navigation via pre-training. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Hausman et al., 2018] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an Embedding Space for Transferable Robot Skills. In *International Conference on Learning Representations (ICLR)*.

[He et al., 2020] He, J., Zhao, L., Yang, H., Zhang, M., and Li, W. (2020). HSI-BERT: Hyperspectral Image Classification Using the Bidirectional Encoder Representation From Transformers. In *IEEE Transactions on Geoscience and Remote Sensing*.

[He et al., 2017] He, K., Gkioxari, G., Dollar, P., and Girshick, R. (2017). Mask R-CNN. In *International Conference on Computer Vision (ICCV)*.

[He et al., 2015] He, K., Lahijanian, M., Kavraki, L., and Vardi, M. (2015). Towards manipulation planning with temporal logic specifications. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Heess et al., 2017] Heess, N., Dhruva, T., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S., Riedmiller,

M. A., and Silver, D. (2017). Emergence of Locomotion Behaviours in Rich Environments. *arXiv preprint*.

[Hennigh et al., 2020] Hennigh, O., Narasimhan, S., Nabian, M., Subramaniam, A., Tangsali, K., Rietmann, M., Ferrandis, J., Byeon, W., Fang, Z., and Choudhry, S. (2020). NVIDIA SimNet: an AI-accelerated multi-physics simulation framework. *arXiv preprint*.

[Hester et al., 2018] Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Sendonaris, A., Dulac-Arnold, G., Osband, I., Agapiou, J., Leibo, J. Z., and Gruslys, A. (2018). Deep Q-learning from Demonstrations. In *Conference on Artificial Intelligence (AAAI)*.

[Ho and Ermon, 2016] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*.

[Hong et al., 2021] Hong, Y., Wu, Q., Qi, Y., Rodriguez-Opazo, C., and Gould, S. (2021). A Recurrent Vision-and-Language BERT for Navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Hosu and Rebedea, 2016] Hosu, I.-A. and Rebedea, T. (2016). Playing atari games with deep reinforcement learning and human checkpoint replay. In *ECAI Workshop on Evaluating General Purpose AI*.

[Huang et al., 2018] Huang, D.-A., Nair, S., Xu, D., Zhu, Y., Garg, A., Fei-Fei, L., Savarese, S., and Niebles, J. C. (2018). Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Huang et al., 2019] Huang, H., Jain, V., Mehta, H., Ku, A., Magalhaes, G., Baldridge, J., and Ie, E. (2019). Transferable representation learning in vision-and-language navigation. In *International Conference on Computer Vision (ICCV)*.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning (ICML)*.

[Irpan, 2018] Irpan, A. (2018). Deep reinforcement learning doesn't work yet. https://www.alexirpan.com/2018/02/14/rl-hard.html.

[Jaderberg et al., 2017] Jaderberg, M., Mnih, V., Czarnecki, W., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations (ICLR)*.

[Jain et al., 2021] Jain, U., Liu, I.-J., Lazebnik, S., Kembhavi, A., Weihs, L., and Schwing, A. (2021). GridToPix: Training Embodied Agents with Minimal Supervision. *arXiv preprint*.

[Jakobi et al., 1995] Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In *European Conference on Artificial Life (ECAL)*.

[James et al., 2018] James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2018). Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Janner et al., 2019] Janner, M., Levine, S., Freeman, W. T., Tenenbaum, J. B., Finn, C., and Wu, J. (2019). Reasoning about physical interactions with object-oriented prediction and planning. In *International Conference on Learning Representations (ICLR)*.

[Karoly et al., 2019] Karoly, A. I., Fuller, R., and Galambos, P. (2019). Unsupervised clustering for deep learning: A tutorial survey. *Acta Polytechnica Hungarica*.

[Karttunen et al., 2020] Karttunen, J., Kanervisto, A., Hautamäki, V., and Kyrki, V. (2020). From video game to real robot: The transfer between action spaces. In *IEEE Conference on Acoustic, Speech, and Signal Processing (ICASSP)*.

[Katharopoulos et al., 2020] Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning (ICML)*.

[Ke et al., 2019] Ke, L., Li, X., Bisk, Y., Holtzman, A., Gan, Z., Liu, J., Gao, J., Choi, Y., and Srinivasa, S. (2019). Tactical rewind: Self-correction via backtracking in vision-and-language navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint*.

[Kober et al., 2013] Kober, J., Bagnell, J., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research (IJRR)*.

[Kolve et al., 2017] Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Gordon, D., Zhu, Y., Gupta, A., and Farhadi, A. (2017). AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv preprint*.

[Kostrikov, 2018] Kostrikov, I. (2018). Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr.

[Krantz et al., 2020] Krantz, J., Wijmans, E., Majumdar, A., Batra, D., and Lee, S. (2020). Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *European Conference on Computer Vision (ECCV)*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Ku et al., 2020] Ku, A., Anderson, P., Patel, R., Ie, E., and Baldridge, J. (2020). Room-Across-Room: Multilingual Vision-and-Language Navigation with Dense Spatiotemporal Grounding. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Kulkarni et al., 2016] Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. (2016). Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Kumar et al., 2016] Kumar, V., Gupta, A., Todorov, E., and Levine, S. (2016). Learning Dexterous Manipulation Policies from Experience and Imitation. *arXiv preprint*.

[Labbe et al., 2020] Labbe, Y., Zagoruyko, S., Kalevatykh, I., Laptev, I., Carpentier, J., Aubry, M., and Sivic, J. (2020). Monte-carlo tree search for efficient visually guided rearrangement planning. *IEEE Robotics and Automation Letters*.

[Lampe and Riedmiller, 2013] Lampe, T. and Riedmiller, M. (2013). Acquiring visual servoing reaching and grasping skills using neural reinforcement learning. In *IEEE International Joint Conference on Neural Networks (IJCNN)*.

[Laskey et al., 2017] Laskey, M., Lee, J., Fox, R., Dragan, A. D., and Goldberg, K. (2017). DART: noise injection for robust imitation learning. In *Conference on Robot Learning (CoRL)*.

[LaValle, 2006] LaValle, S. (2006). *Planning Algorithms*. Cambridge University.

[Le et al., 2018] Le, H. M., Jiang, N., Agarwal, A., Dudík, M., Yue, Y., and III, H. D. (2018). Hierarchical Imitation and Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

[LeCun et al., 1989] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*.

[Lee et al., 2018] Lee, K.-H., Ros, G., Li, J., and Gaidon, A. (2018). SPIGAN: Privileged Adversarial Learning from Simulation. In *International Conference on Learning Representations (ICLR)*.

[Lee et al., 2019] Lee, Y., Sun, S.-H., Somasundaram, S., S. Hu, E., and J. Lim, J. (2019). Composing Complex Skills by Learning Transition Policies with Proximity Reward Induction. In *International Conference on Learning Representations (ICLR)*.

[Levine et al., 2015] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2015). End-to-End Training of Deep Visuomotor Policies. *The Journal of Machine Learning Research*.

[Levine et al., 2016] Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *International Society for Engineers and Researchers (ISER)*.

[Li et al., 2020] Li, J., Wang, X., Tang, S., Shi, H., Wu, F., Zhuang, Y., and Wang, W. Y. (2020). Unsupervised Reinforcement Learning of Transferable Meta-Skills for Embodied Navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Liang et al., 2018] Liang, J., Makoviychuk, V., Handa, A., Chentanez, N., Macklin, M., and Fox, D. (2018). Gpu-accelerated robotic simulation for distributed reinforcement learning. In *Conference on Robot Learning (CoRL)*.

[Lillicrap et al., 2016] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[Litvak et al., 2019] Litvak, Y., Biess, A., and Bar-Hillel, A. (2019). Learning a High-Precision Robotic Assembly Task Using Pose Estimation from Simulated Depth Images. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Liu et al., 2019] Liu, H., Zhang, Z., Xie, X., Zhu, Y., Liu, Y., Wang, Y., and Zhu, S. (2019). High-fidelity grasping in virtual reality using a glove-based system. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Loshchilov and Hutter, 2019] Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

[Lozano-Pérez and Kaelbling, 2014] Lozano-Pérez, T. and Kaelbling, L. P. (2014). A constraint-based method for solving sequential manipulation planning problems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Lu et al., 2019] Lu, J., Batra, D., Parikh, D., and Lee, S. (2019). Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Lynch et al., 2019] Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. (2019). Learning latent plans from play. In *Conference on Robot Learning (CoRL)*.

[Lynch and Sermanet, 2020] Lynch, C. and Sermanet, P. (2020). Grounding Language in Play. *arXiv preprint*.

[Ma et al., 2019a] Ma, C.-Y., Lu, J., Wu, Z., AlRegib, G., Kira, Z., Socher, R., and Xiong, C. (2019a). Self-Monitoring Navigation Agent via Auxiliary Progress Estimation. In *International Conference on Learning Representations (ICLR)*.

[Ma et al., 2019b] Ma, C.-Y., Wu, Z., AlRegib, G., Xiong, C., and Kira, Z. (2019b). The regretful agent: Heuristic-aided navigation through progress estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[MacMahon et al., 2006] MacMahon, M., Stankiewicz, B., and Kuipers, B. (2006). Walk the talk: Connecting language, knowledge, and action in route instructions. In *Conference on Artificial Intelligence (AAAI)*.

[Mahler et al., 2017] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. Y. (2017). Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Mahmood et al., 2018] Mahmood, A., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning (CoRL)*.

[Majumdar et al., 2020] Majumdar, A., Shrivastava, A., Lee, S., Anderson, P., Parikh, D., and Batra, D. (2020). Improving vision-and-language navigation with image-text pairs from the web. In *European Conference on Computer Vision (ECCV)*.

[Manna and Pnueli, 1992] Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, Berlin, Heidelberg.

[Mehta et al., 2020] Mehta, H., Artzi, Y., Baldridge, J., Ie, E., and Mirowski, P. (2020). Retouchdown: Adding touchdown to streetlearn as a shareable resource for language grounding tasks in street view. *arXiv preprint*.

[Mei et al., 2016] Mei, H., Bansal, M., and Walter, M. (2016). Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Conference on Artificial Intelligence (AAAI)*.

[Michie et al., 1990] Michie, D., Bain, M., and Hayes-Michie, J. (1990). Cognitive models from subcognitive skills. *IEEE Control Engineering Series*.

[Misra et al., 2017] Misra, D., Langford, J., and Artzi, Y. (2017). Mapping instructions and visual observations to actions with reinforcement learning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Misra et al., 2016] Misra, D. K., Sung, J., Lee, K., and Saxena, A. (2016). Tell me dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari With Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS) Deep Learning Workshop*.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*.

[Morales and Sammut, 2004] Morales, E. F. and Sammut, C. (2004). Learning to fly by combining reinforcement learning with behavioural cloning. In *International Conference on Machine Learning (ICML)*.

[Mordatch et al., 2016] Mordatch, I., Mishra, N., Eppner, C., and Abbeel, P. (2016). Combining model-based policy search with online model learning for control of physical humanoids. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Mueller et al., 2018] Mueller, M., Dosovitskiy, A., Ghanem, B., and Koltun, V. (2018). Driving policy transfer via modularity and abstraction. In *Conference on Robot Learning (CoRL)*.

[Munos et al., 2016] Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. G. (2016). Safe and Efficient Off-Policy Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Nachum et al., 2018] Nachum, O., Gu, S., Lee, H., and Levine, S. (2018). Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Nachum et al., 2019] Nachum, O., Tang, H., Lu, X., Gu, S., Lee, H., and Levine, S. (2019). Why does hierarchy (sometimes) work so well in reinforcement learning? *Advances in Neural Information Processing Systems (NeurIPS)*.

[Nagabandi et al., 2019] Nagabandi, A., Clavera, I., Liu, S., Fearing, R., Abbeel, P., Levine, S., and Finn, C. (2019). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

[Nair et al., 2018] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Nair et al., 2020] Nair, S., Babaeizadeh, M., Finn, C., Levine, S., and Kumar, V. (2020). Time Reversal as Self-Supervision. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Narasimhan et al., 2020] Narasimhan, M., Wijmans, E., Chen, X., Darrell, T., Batra, D., Parikh, D., and Singh, A. (2020). Seeing the Un-Scene: Learning Amodal Semantic Maps for Room Navigation. In *European Conference on Computer Vision (ECCV)*.

[Newell et al., 2016] Newell, A., Yang, K., and Deng, J. (2016). Stacked hourglass networks for human pose estimation. In *European Conference on Computer Vision (ECCV)*.

[Ng et al., 1999] Ng, A., Harada, D., and Russell, S. J. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning (ICML)*.

[Ng and Russell, 2000] Ng, A. Y. and Russell, S. J. (2000). Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*.

[OpenAI, 2018] OpenAI (2018). Openai five. https://blog.openai.com/openai-five/.

[Pan et al., 2018] Pan, Y., Cheng, C.-A., Saigol, K., Lee, K., Yan, X., Theodorou, E., and Boots, B. (2018). Agile Autonomous Driving using End-to-End Deep Imitation Learning. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Parisotto et al., 2020] Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M., Heess, N., and Hadsell, R. (2020). Stabilizing Transformers for Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

[Pashevich et al., 2018] Pashevich, A., Hafner, D., Davidson, J., Sukthankar, R., and Schmid, C. (2018). Modulated Policy Hierarchies. In *Advances in Neural Information Processing Systems (NeurIPS) Deep Reinforcement Learning workshop*.

[Pashevich et al., 2020a] Pashevich, A., Kalevatykh, I., Laptev, I., and Schmid, C. (2020a). Learning visual policies for building 3D shape categories. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Pashevich et al., 2020b] Pashevich, A., Kalevatykh, I., Laptev, I., and Schmid, C. (2020b). Learning visual policies for building 3D shape categories, project webpage . http://pascal.inrialpes.fr/data2/3D-shapes/.

[Pashevich et al., 2021a] Pashevich, A., Schmid, C., and Sun, C. (2021a). Episodic Transformer for Vision-and-Language Navigation. *arXiv preprint*.

[Pashevich et al., 2021b] Pashevich, A., Schmid, C., and Sun, C. (2021b). Episodic Transformer for Vision-and-Language Navigation, project webpage. https://sites.google.com/view/episodictransformer.

[Pashevich et al., 2019a] Pashevich, A., Strudel, R., Kalevatykh, I., Laptev, I., and Schmid, C. (2019a). Learning to augment synthetic images for sim2real policy transfer. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Pashevich et al., 2019b] Pashevich, A., Strudel, R., Kalevatykh, I., Laptev, I., and Schmid, C. (2019b). Learning to augment synthetic images for sim2real policy transfer, project webpage. http://pascal.inrialpes.fr/data2/sim2real.

[Patel et al., 2020] Patel, R., Pavlick, E., and Tellex, S. (2020). Grounding Language to Non-Markovian Tasks with No Supervision of Task Specifications. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Patel et al., 2019] Patel, R., Pavlick, R., and Tellex, S. (2019). Learning to ground language to temporal logical form. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

[Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*.

[Paul et al., 2018] Paul, R., Arkin, J., Aksaray, D., Roy, N., and Howard, T. M. (2018). Efficient grounding of abstract spatial concepts for natural language interaction with robot platforms. *The International Journal of Robotics Research*.

[Paulin et al., 2014] Paulin, M., Revaud, J., Harchaoui, Z., Perronnin, F., and Schmid, C. (2014). Transformation pursuit for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Paxton et al., 2019] Paxton, C., Barnoy, Y., Katyal, K., Arora, R., and Hager, G. D. (2019). Visual robot task planning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Peng et al., 2016] Peng, X. B., Berseth, G., and van de Panne, M. (2016). Terrain-adaptive locomotion skills using deep reinforcement learning. *Transactions on Graphics*.

[Perez et al., 2018] Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. C. (2018). FiLM: Visual Reasoning with a General Conditioning Layer. In *Conference on Artificial Intelligence (AAAI)*.

[Pinto et al., 2018] Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. (2018). Asymmetric Actor Critic for Image-Based Robot Learning. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Pinto and Gupta, 2016] Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Plappert et al., 2018] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint*.

[Pomerleau, 1989] Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Popov et al., 2017] Popov, I., Heess, N., Lillicrap, T., Hafner, R., Barth-Maron, G., Vecerík, M., Lampe, T., Tassa, Y., Erez, T., and Riedmiller, M. (2017). Data-efficient Deep Reinforcement Learning for Dexterous Manipulation. *arXiv preprint*.

[Puig et al., 2018] Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., and Torralba, A. (2018). Virtualhome: Simulating household activities via programs. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Rahmatizadeh et al., 2018] Rahmatizadeh, R., Abolghasemi, P., Bölöni, L., and Levine, S. (2018). Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-to-End Learning from Demonstration. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Ratliff et al., 2008] Ratliff, N., Bagnell, J. A., and Srinivasa, S. S. (2008). Imitation learning for locomotion and manipulation. In *IEEE Robotics and Automation Society (RAS)*.

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Riedmiller et al., 2018] Riedmiller, M. A., Hafner, R., Lampe, T., Neunert, M., Degrave, J., de Wiele, T. V., Mnih, V., Heess, N., and Springenberg, J. T. (2018). Learning by playing - solving sparse reward tasks from scratch. In *International Conference on Machine Learning (ICML)*.

[Rosenblatt, 1957] Rosenblatt, F. (1957). The perceptron - a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory.

[Ross and Bagnell, 2014] Ross, S. and Bagnell, J. A. (2014). Reinforcement and Imitation Learning via Interactive No-Regret Learning. *arXiv preprint*.

[Ruiz et al., 2019] Ruiz, N., Schulter, S., and Chandraker, M. (2019). Learning To Simulate. In *International Conference on Learning Representations (ICLR)*.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*.

[Sadeghi and Levine, 2017] Sadeghi, F. and Levine, S. (2017). CAD2RL: Real Single-Image Flight Without a Single Real Image. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Sadeghi et al., 2018] Sadeghi, F., Toshev, A., Jang, E., and Levine, S. (2018). Sim2Real View Invariant Visual Servoing by Recurrent Control. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Savva et al., 2019] Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., et al. (2019). Habitat: A platform for embodied ai research. In *International Conference on Computer Vision (ICCV)*.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint*.

[Shelhamer et al., 2017] Shelhamer, E., Long, J., and Darrell, T. (2017). Fully Convolutional Networks for Semantic Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

[Shridhar et al., 2020] Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. (2020). ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Shridhar et al., 2021] Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. (2021). Alfworld: Aligning text and embodied environments for interactive learning. *International Conference on Learning Representations (ICLR)*.

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L. R., Lai, M., Bolton, A., Chen, Y., Lillicrap, T. P., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*.

[Simonyan and Zisserman, 2014a] Simonyan, K. and Zisserman, A. (2014a). Two-Stream Convolutional Networks for Action Recognition in Videos. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Simonyan and Zisserman, 2014b] Simonyan, K. and Zisserman, A. (2014b). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint*.

[Singh et al., 2020] Singh, K. P., Bhambri, S., Kim, B., Mottaghi, R., and Choi, J. (2020). MOCA: A Modular Object-Centric Approach for Interactive Instruction Following. *arXiv preprint*.

[Srivastava et al., 2014] Srivastava, S., Fang, E., Riano, L., Chitnis, R., Russell, S., and Abbeel, P. (2014). Combined task and motion planning through an extensible planner-independent interface layer. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Stepputtis et al., 2020] Stepputtis, S., Campbell, J., Phielipp, M., Lee, S., Baral, C., and Amor, H. B. (2020). Language-conditioned imitation learning for robot manipulation tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Strudel et al., 2020a] Strudel, R., Pashevich, A., Kalevatykh, I., Laptev, I., and Schmid, C. (2020a). Learning to combine primitive skills: A step towards versatile robotic manipulation, project webpage. https://www.di.ens.fr/willow/research/rlbc/.

[Strudel et al., 2020b] Strudel, R., Pashevich, A., Kalevatykh, I., Laptev, I., Sivic, J., and Schmid, C. (2020b). Learning to combine primitive skills: A step towards versatile robotic manipulation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Suárez-Ruiz et al., 2018] Suárez-Ruiz, F., Zhou, X., and Pham, Q.-C. (2018). Can robots assemble an ikea chair? *Science Robotics*.

[Şucan et al., 2012] Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *Robotics and Automation Magazine*.

[Sukhbaatar et al., 2017] Sukhbaatar, S., Kostrikov, I., Szlam, A., and Fergus, R. (2017). Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations (ICLR)*.

[Sun et al., 2019] Sun, C., Myers, A., Vondrick, C., Murphy, K., and Schmid, C. (2019). VideoBERT: A Joint Model for Video and Language Representation Learning. In *International Conference on Computer Vision (ICCV)*.

[Sun et al., 2017] Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting Unreasonable Effectiveness of Data in Deep Learning Era. In *International Conference on Computer Vision (ICCV)*.

[Sun et al., 2018] Sun, W., Bagnell, J. A., and Boots, B. (2018). Truncated horizon policy search: Combining reinforcement learning & imitation learning. In *International Conference on Learning Representations (ICLR)*.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). Reinforcement Learning: An Introduction. *The MIT Press*.

[Sutton et al., 1999] Sutton, R. S., Precup, D., and Singh, S. (1999). Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*.

[Tan et al., 2019] Tan, H., Yu, L., and Bansal, M. (2019). Learning to Navigate Unseen Environments: Back Translation with Environmental Dropout. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

[Tan et al., 2018] Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. (2018). Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Tang et al., 2017] Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F., and Abbeel, P. (2017). Exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Tellex et al., 2011] Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., Teller, S., and Roy, N. (2011). Understanding natural language commands for robotic navigation and mobile manipulation. In *Conference on Artificial Intelligence (AAAI)*.

[Tenorth et al., 2010] Tenorth, M., Nyga, D., and Beetz, M. (2010). Understanding and executing instructions for everyday manipulation tasks from the world wide web. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Tobin et al., 2017] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Tobin et al., 2018] Tobin, J., Zaremba, W., and Abbeel, P. (2018). Domain Randomization and Generative Models for Robotic Grasping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

[Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*.

[Toussaint, 2015] Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[Trott et al., 2019] Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Van-Quang Nguyen, 2020] Van-Quang Nguyen, T. O. (2020). A hierarchical attention model for action learning from realistic environments and

directives. In *European Conference on Computer Vision (ECCV) EVAL Workshop*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Vecerik et al., 2017] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothorl, T., Lampe, T., and Riedmiller, M. A. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint*.

[Vezhnevets et al., 2017] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). FeUdal Networks for Hierarchical Reinforcement Learning. In *International Conference on Machine Learning (ICML)*.

[Vinyals et al., 2015] Vinyals, O., Kaiser, L., Koo, T., Petrov, S., Sutskever, I., and Hinton, G. (2015). Grammar as a Foreign Language. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Wang et al., 2019a] Wang, A., Kurutach, T., Liu, K., Abbeel, P., and Tamar, A. (2019a). Learning Robotic Manipulation through Visual Planning and Acting. *arXiv preprint*.

[Wang et al., 2018] Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Wang et al., 2019b] Wang, X., Huang, Q., Asli, C., Gao, J., Shen, D., Wang, Y.-F., Wang, W., and Zhang, L. (2019b). Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Wei et al., 2016] Wei, S.-E., Ramakrishna, V., Kanade, T., and Sheikh, Y. (2016). Convolutional Pose Machines. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Łukasz Kaiser, Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016). Google's neural machine translation

system: Bridging the gap between human and machine translation. *arXiv preprint*.

[Wu et al., 2015] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Xu et al., 2020] Xu, T., Li, Z., and Yu, Y. (2020). Error bounds of imitating policies and environments. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[Yang et al., 2012] Yang, N., Kim, Y., and Park, R. (2012). Depth hole filling using the depth distribution of neighboring regions of depth holes in the Kinect sensor. In *IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*.

[Young et al., 2018] Young, T., Hazarika, D., Poria, S., and Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*.

[Yu et al., 2018] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *Conference on Empirical Methods in Natural Language Processing*.

[Zaheer et al., 2020] Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. (2020). Big Bird: Transformers for Longer Sequences. *Advances in Neural Information Processing Systems (NeurIPS)*.

[Zakka et al., 2019] Zakka, K., Zeng, A., Lee, J., and Song, S. (2019). Form2Fit: Learning Shape Priors for Generalizable Assembly from Disassembly. *IEEE International Conference on Robotics and Automation (ICRA)*.

[Zelle and Mooney, 1996] Zelle, J. M. and Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. In *Conference on Artificial Intelligence (AAAI)*.

[Zeng et al., 2019] Zeng, A., Song, S., Lee, J., Rodríguez, A., and Funkhouser, T. A. (2019). TossingBot: Learning to Throw Arbitrary Objects with Residual Physics. In *Proceedings of Robotics: Science and Systems (RSS)*.

[Zettlemoyer and Collins, 2007] Zettlemoyer, L. and Collins, M. (2007). Online learning of relaxed ccg grammars for parsing to logical form. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[Zettlemoyer and Collins, 2005] Zettlemoyer, L. S. and Collins, M. (2005). Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *Conference on Uncertainty in Artificial Intelligence (UAI)*.

[Zhang et al., 2018] Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K. Y., and Abbeel, P. (2018). Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Zhong et al., 2017] Zhong, Y., Dai, Y., and Li, H. (2017). Self-supervised learning for stereo matching with self-improving ability. *arXiv preprint*.

[Zhu et al., 2020] Zhu, F., Zhu, Y., Chang, X., and Liang, X. (2020). Vision-language navigation with self-supervised auxiliary reasoning tasks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[Zhu et al., 2017] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017). Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[Zhu et al., 2018] Zhu, Y., Wang, Z., Merel, J., Rusu, A. A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., and Heess, N. (2018). Reinforcement and imitation learning for diverse visuomotor skills. *Proceedings of Robotics: Science and Systems (RSS)*.