# Deep Reinforcement Learning with Prior Knowledge

Tao Chen

CMU-RI-TR-19-09

May 2019


The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Thesis Committee:**
Abhinav Gupta, *chair*
Oliver Kroemer
Adithyavairavan Murali

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

# Abstract

Deep reinforcement learning has been applied to many domains from computer games, natural language processing, recommendation systems to robotics. While there are many scenarios where huge amounts of data is easily available such as games, the applications of deep reinforcement learning to robotics is often limited by the bottleneck of acquiring data. Hence, generalization becomes essential in making the learning algorithm practical in robotics. We found out that using prior knowledge of the tasks can significantly boost the learning performance and generalization capabilities.

Deep reinforcement learning could be used to learn dexterous robotic policies but it is challenging to transfer them to new robots with vastly different hardware properties. It is also prohibitively expensive to learn a new policy from scratch for each robot hardware due to the high sample complexity of modern state-of-the-art algorithms. We propose a novel approach called *Hardware Conditioned Policies* where we train a universal policy conditioned on a vector representation of robot hardware. We considered robots in simulation with varied dynamics, kinematic structure, kinematic lengths and degrees-of-freedom and show better generalization with our method.

In this thesis, we also explore the generalization problem in navigation. Even though numerous past works have tackled the problem of task-driven navigation, how to effectively explore a new environment to enable a variety of down-stream tasks has received much less attention. We study how agents can autonomously explore realistic and complex 3D environments without the context of task-rewards. We propose a learning-based approach and investigate different policy architectures, reward functions, and training paradigms. We find that use of policies with spatial memory that are bootstrapped with imitation learning and finally fine-tuned with coverage rewards derived purely from on-board sensors can be effective at exploring novel environments. We also show how such task-agnostic exploration can be used for down-stream tasks.

## Acknowledgments

First, I would like to thank my advisor, Professor Abhinav Gupta, for all his support throughout my masters program. I am grateful for having many helpful discussions with him and his unique insights had deep impact on my research. I would also like to thank Professor Oliver Kroemer for his constructive feedback and suggestions on my work. I am thankful for Professor Saurabh Gupta and Adithya Murali's help and advice on my projects. Futhermore, I would like to thank all my lab colleagues for the help with the research and life.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

A long-lasting goal of robotics is to perceive the world, reason the consequence, and react to the changes. One of the fundamental challenges involved here is the sequential decision making problem. An intelligent machine needs to have the ability to predict the future and make long-horizon sequential decisions in order to interact coherently with the environment. Reinforcement learning is a learning framework which formally defines and provides mathematical tools to approach such problems. One of the most notable formula used in reinforcement learning is the Bellman equation, which converts the dynamic programming problem into a sequence of sub-problems and makes the optimization tractable. To make reinforcement learning algorithms applicable to high-dimensional state spaces and action spaces, researchers often use high-capacity non-linear functions, such as deep neural networks, to approximate policy or value functions. We have seen many such successful applications in computer games and robotics control [1, 2, 3]. However, this brings the downside of high demands of big data, which is often quite expensive and infeasible to collect in real world robotic settings. Therefore, more data-efficient or more transferable learning policies are desired.

The state-of-the-art model-free deep reinforcement learning algorithms [4, 5, 6] provide a promising approach in solving sequential decision making problems without the knowledge of the environment model (dynamics). This make the learning algorithms more general and applicable to various domains. However, the huge amount of data needed to train such algorithms make it extremely expensive to be

re-trained from scratch in order to be applied to different agents or environments. This thesis focuses on how to equip deep reinforcement learning algorithms with better generalization capabilities so that it reduces the demands of big data when the policy is applied to a new agent or a new environment. More specifically, we looked into two fields: multi-robot transfer learning and navigation.

Even though we have seen remarkable progress in deep reinforcement learning, one major shortcoming of the current approaches is that the policy learned for a robot is specific to the hardware, and thus tends to over-fit to one robot. This prevents the policy from generalizing to other robots. So we propose a method called Hardware Conditioned Policies to improve the policy generalization over robots with different kinematics structures, link length and dynamics. The main idea is to augment the state with an additional robot-specific representation using prior knowledge to make the policy aware of the robot characteristics. Such a representation can be either explicitly constructed given enough prior knowledge or implicitly learned via back-propagation. The experiments show that our methods greatly improve the generalization of the policy over different robots.

In navigation, there have been many works formulating the navigation as a learning problem [7, 8, 9, 10] in the past few years. Learning methods are poised to learn the semantic structures and common sense of the environment and thus have the promises to overcome the requirement of precise depth sensors required by most of the geometric-based navigation methods such as SLAM. We developed a learning method that can learn an exploration policy which generalizes to completely new environments. The goal of the policy is to make the agent automatically explore the environment when it is deployed in a new house or building, similar to what a human does when he/she walks into a new building. We propose a novel policy architecture with an augmented state representation using prior knowledge. Our results show that our policy can generalize to new environments much better compared to the policy without the state augmentation.

We show the detailed alogrithms and experiments in multi-robot transfer learning in Chapter 2 and navigation in Chapter 3.

# Chapter 2

# Multi-Robot Transfer Learning

## 2.1 Introduction

In recent years, we have seen remarkable success in the field of deep reinforcement learning (DRL). From learning policies for games [11, 12] to training robots in simulators [5], neural network based policies have shown remarkable success. But will these successes translate to real world robots? Can we use DRL for learning policies of how to open a bottle, grasping or even simpler tasks like fixturing and peg-insertion? One major shortcoming of current approaches is that they are not sample-efficient. We need millions of training examples to learn policies for even simple actions. Another major bottleneck is that these policies are specific to the hardware on which training is performed. If we apply a policy trained on one robot to a different robot it will fail to generalize. Therefore, in this paradigm, one would need to collect millions of examples for each task and each robot.

But what makes this problem even more frustrating is that since there is no standardization in terms of hardware, different labs collect large-scale data using different hardware. These hardware vary in degrees of freedom (DOF), kinematic design and even dynamics. Because the learning process is so hardware-specific, there is no way to pool and use all the shared data collected across using different types of robots, especially when the robots are trained under torque control. There have been efforts to overcome dependence on hardware properties by learning invariance to robot dynamics using dynamic randomization [13]. However, learning a policy

invariant to other hardware properties such as degrees of freedom and kinematic structure is a challenging problem.

In this chapter, we propose an alternative solution: instead of trying to learn the invariance to hardware; we embrace these differences and propose to learn a policy conditioned on the hardware properties itself. Our core idea is to formulate the policy $\pi$ as a function of current state $s_t$ and the hardware properties $v_h$. So, in our formulation, the policy decides the action based on current state and its own capabilities (as defined by hardware vector). But how do you represent the robot hardware as vector? In this paper, we propose two different possibilities. First, we propose an explicit representation where the kinematic structure itself is fed as input to the policy function. But such an approach will not be able to encode robot dynamics which might be hard to measure. Therefore, our second solution is to learn an embedding space for robot hardware itself. Our results indicate that encoding the kinematic structures explicitly enables high success rate on zero-shot transfer to new kinematic structure. And learning the embedding vector for hardware implicitly without using kinematics and dynamics information is able to give comparable performance to the model where we use all of the kinematics and dynamics information. Finally, we also demonstrate that the learned policy can also adapt to new robots with much less data samples via finetuning.

## 2.2 Related Work

**Transfer in Robot Learning** Transfer learning has a lot of practical value in robotics, given that it is computationally expensive to collect data on real robot hardware and that many reinforcement learning algorithms have high sample complexity. Taylor et al. present an extensive survey of different transfer learning work in reinforcement learning [14]. Prior work has broadly focused on the transfer of policies between tasks [15, 16, 17, 18], control parameters [19], dynamics [13, 20, 21, 22], visual inputs [23], non-stationary environments [24], goal targets [25]. Nilim et. al. presented theoretical results on the performance of transfer under conditions with bounded disturbances in the dynamics [21]. There have been efforts in applying domain adaption such as learning common invariant feature spaces between domains [26] and learning a mapping from target to source domain [27]. Such approaches

require prior knowledge and data from the target domain. A lot of recent work has focused on transferring policies trained in simulation to a real robot [13, 23, 28, 29]. However, there has been very limited work on transferring knowledge and skills between different robots [15, 30]. The most relevant paper is Devin et al. [15], who propose module networks for transfer learning and used it to transfer 2D planar policies across hand-designed robots. The key idea is to decompose policy network into robot-specific module and task-specific module. In our work, a universal policy is conditioned on a vector representation of the robot hardware - the policy does not necessarily need to be completely retrained for a new robot. There has also been some concurrent work in applying graph neural networks (GNN) as the policy class in continuous control [31, 32]. [31] uses a GNN instead of a MLP to train a policy. [32] uses a GNN to learn a forward prediction model for future states and performs model predictive control on agents. Our work is orthogonal to these methods as we condition the policy on the augmented state of the robot hardware, which is independant of the policy class.

**Robust Control and Adaptive Control** Robust control can be considered from several vantage points. In the context of trajectory optimization methods, model predictive control (MPC) is a popular framework which continuously resolves an open-loop optimization problem, resulting in a closed loop algorithm that is robust to dynamics disturbances. In the context of deep reinforcement learning, prior work has explored trajectory planning with an ensemble of dynamics models [33], adversarial disturbances [22], training with random dynamics parameters in simulation [13, 20], etc. [13] uses randomization over dynamics so that the policy network can generalize over a large range of dynamics variations for both the robot and the environment. However, it uses position control where robot dynamics has little direct impact on control. We use low-level torque control which is severely affected by robot dynamics and show transfer even between kinematically different agents. There have been similar works in the area of adaptive control [34] as well, where unknown dynamics parameters are estimated online and adaptive controllers adapt the control parameters by tracking motion error. Our work is a model-free method which does not make assumptions like linear system dynamics. We also show transfer results on robots with different DOFs and joint displacements.

**System Identification** System identification is a necessary process in robotics to

find unknown physical parameters or to address model inconsistency during training and execution. For control systems based on analytic models, as is common in the legged locomotion community, physical parameters such as the moment of inertia or friction have to be estimated for each custom robotic hardware [35, 36]. Another form of system identification involves the learning of a dynamics model for use in model-based reinforcement learning. Several prior research work have iterated between building a dynamics model and policy optimization [37, 38, 39]. In the context of model-free RL, Yu et al. proposed an Online System Identification [40] module that is trained to predict physical environmental factors such as the agent mass, friction of the floor, etc. which are then fed into the policy along with the agent state [40]. However, results were shown for simple simulated domains and even then it required a lot of samples to learn an accurate regression function of the environmental factors. There is also concurrent work which uses graph networks [32] to learn a forward prediction model for future states and to perform model predictive control. Our method is model-free and only requires a simple hardware augmentation as input regardless of the policy class or DRL algorithms.

## 2.3 Preliminaries

We consider the multi-robot transfer learning problem under the reinforcement learning framework and deal with fully observable environments that are modeled as continuous space Markov Decision Processes (MDP). The MDPs are represented by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$, where $\mathcal{S}$ is a set of continuous states, $\mathcal{A}$ is a set of continuous actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $\rho_0$ is the initial state distribution, and $\gamma \in (0, 1]$ is the discount factor. The aim is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected return.

There are two classes of approaches used for optimization: on-policy and off-policy. On-policy approaches (e.g., Proximal Policy Optimization (PPO) [4]) optimize the same policy that is used to make decisions during exploration. On the other hand, off-policy approaches allow policy optimization on data obtained by a behavior policy different from the policy being optimized. Deep deterministic policy gradient (DDPG) [5] is a model-free actor-critic off-policy algorithm which uses deterministic action

policy and is applicable to continuous action spaces.

One common issue with training these approaches is sparse rewards. Hindsight experience replay (HER) [41] was proposed to improve the learning under the sparse reward setting for off-policy algorithms. The key insight of HER is that even though the agent has not succeeded at reaching the specified goal, the agent could have at least achieved a different one. So HER pretends that the agent was asked to achieve the goal it ended up with in that episode at the first place, instead of the one that we set out to achieve originally. By repeating the goal substitution process, the agent eventually learns how to achieve the goals we specified.

## 2.4 Hardware Conditioned Policies

Our proposed method, *Hardware Conditioned Policies (HCP)*, takes robot hardware information into account in order to generalize the policy network over robots with different kinematics and dynamics. The main idea is to construct a vector representation $v_h$ of each robot hardware that can guide the policy network to make decisions based on the hardware characteristics. Therefore, the learned policy network should learn to act in the environment, conditioned on both the state $s_t$ and $v_h$. There are several factors that encompass robot hardware that we have considered in our framework - robot kinematics (*degree of freedom*, *kinematic structure* such as relative joint positions and orientations, and link length), robot dynamics (*joint damping*, *friction*, *armature*, and *link mass*) - and other aspects such as shape geometry, actuation design, etc. that we will explore in future work. It is also noteworthy that the robot kinematics is typically available for any newly designed robot, for instance through the Universal Robot Description Format-URDF [42]. Nonetheless, the dynamics are typically not available and may be inaccurate or change over time even if provided. We now explain two ways on how to encode the robot hardware via vector $v_h$.

### 2.4.1 Explicit Encoding

First, we propose to represent robot hardware information via an explicit encoding method (HCP-E). In explicit encoding, we directly use the kinematic structure as input to the policy function. Note that while estimating the kinematic structure is

feasible, it is not feasible to measure dynamics. However, some environments and tasks might not be heavily dependent on robot dynamics and in those scenarios explicit encoding (HCP-E) might be simple and more practical than implicit embedding[1]. We followed the popular URDF convention in ROS to frame our explicit encoding and incorporate the least amount of information to fully define a multi-DOF robot for the explicit encoding. It is difficult to completely define a robot with just its end-effector information, as the kinematic structure (even for the same DOF) affects the robot behaviour. For instance, the whole kinematic chain is important when there are obstacles in the work space and the policy has to learn to avoid collisions with its links.

We consider manipulators composed of $n$ revolute joints $(J_0, J_1, ..., J_{n-1})$. Figure 2.1 shows two consecutive joints $J_i, J_{i+1}$ on the two ends of an L-shape link and their corresponding local coordinate systems $\{x_i y_i z_i\}$ with origin $O_i$ and $\{x_{i+1} y_{i+1} z_{i+1}\}$ with origin $O_{i+1}$ where $z$-axis is the direction of revolute joint axis. To represent spatial relationship between $J_i, J_{i+1}$, one needs to know the relative pose[2] $P_i$ between $J_i$ and $J_{i+1}$.

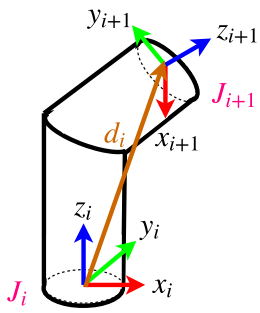



**Figure 2.1:** Local coordinate systems for two consecutive joints

Relative pose $P_i$ can be decomposed into relative position and orientation. Relative position is represented by the difference vector $d_i$ between $O_i$ and $O_{i+1}$, i.e., $d_i = O_{i+1} - O_i$ and $d_i \in \mathbb{R}^3$. The relative rotation matrix from $\{x_{i+1} y_{i+1} z_{i+1}\}$ to $\{x_i y_i z_i\}$ is $\mathbf{R}_i^{i+1} = (\mathbf{R}_w^i)^{-1} \mathbf{R}_w^{i+1}$, where $\mathbf{R}_w^i$ is the rotation matrix of $\{x_i y_i z_i\}$ relative to the world coordinate system. One can further convert rotation matrix which has 9 elements into Euler rotation vector with only 3 independent elements. Therefore, relative rotation can be represented by an Euler rotation vector $e_i = (\theta_{ix}, \theta_{iy}, \theta_{iz}), e_i \in \mathbb{R}^3$. The relative pose is then $P_i = d_i \oplus e_i \in \mathbb{R}^6$, where $\oplus$ denotes concatenation.

With relative pose $P_i$ of consecutive joints in hand, the encoding vector $v_h$ to

[1] We will elaborate such environments on section 2.5.1. We also experimentally show the effect of dynamics for transferring policies in such environments in Appendix A.3.1 and A.3.2.

[2] If the manipulators only differ in length of all links, $v_h$ can be simply the vector of each link's length. When the kinematic structure and DOF also vary, $v_h$ composed of link length is not enough.

represent the robot can be explicitly constructed as follows[3]:

$$v_h = P_{-1} \oplus P_0 \oplus \cdots \oplus P_{n-1}$$

### 2.4.2 Implicit Encoding

In the above section, we discussed how kinematic structure of the hardware can be explicitly encoded as $v_h$. However, in most cases, we need to not only encode kinematic structure but also the underlying dynamic factors. In such scenarios, explicit encoding is not possible since one cannot measure friction or damping in motors so easily. In this section, we discuss how we can learn an embedding space for robot hardware while simultaneously learning the action policies. Our goal is to estimate $v_h$ for each robot hardware such that when a policy function $\pi(s_t, v_h)$ is used to take actions it maximizes the expected return. For each robot hardware, we initialize $v_h$ randomly. We also randomly initialize the parameters of the policy network. We then use standard policy optimization algorithms to update network parameters via back-propagation. However, since $v_h$ is also a learned parameter, the gradients flow back all the way to the encoding vector $v_h$ and update $v_h$ via gradient descent: $v_h \leftarrow v_h - \alpha \nabla_{v_h} L(v_h, \theta)$, where $L$ is the cost function, $\alpha$ is the learning rate. Intuitively, HPC-I trains the policy $\pi(s_t, v_h)$ such that it not only learns a mapping from states to actions that maximizes the expected return, but also finds a good representation for the robot hardware simultaneously.

### 2.4.3 Algorithm

The hardware representation vector $v_h$ can be incorporated into many deep reinforcement learning algorithms by augmenting states to be: $\hat{s}_t \leftarrow s_t \oplus v_h$. We use PPO in environments with dense reward and DDPG + HER in environments with sparse reward in this paper. During training, a robot will be randomly sampled in each episode from a pre-generated robot pool $\mathcal{P}$ filled with a large number of robots with different kinematics and dynamics. Alg. 1 provides an overview of our algorithm.

[3] $P_i$ is relative pose from $U$ to $V$. If $i = -1$, $U = J_0$, $V =$ robot base. If $i = 0, 1, ..., n-2$, $U = J_{i+1}$,V= $J_i$. If $i = n-1$, $U =$ end effector, $V = J_{n-1}$.

**Algorithm 1** Hardware Conditioned Policies (HCP)

Initialize a RL algorithm $\Psi$ ▷ e.g. PPO, DDPG, DDPG+HER
Initialize a robot pool $\mathcal{P}$ of size $N$ with robots in different kinematics and dynamics
**for** episode = 1:M **do**
  Sample a robot instance $\mathcal{I} \in \mathcal{P}$
  Sample an initial state $s_0$
  Retrieve the robot hardware representation vector $v_h$
  Run policy $\pi$ in the environment for $T$ timesteps
  Augment all states with $v_h$: $\hat{s} \leftarrow s \oplus v_h$ ▷ $\oplus$ denotes concatenation
  **for** n=1:W **do**
    Optimize actor and critic networks with $\Psi$ via minibatch gradient descent
    **if** $v_h$ is to be learned (i.e. for implicit encoding, HCP-I) **then**
      update $v_h$ via gradient descent in the optimization step as well
    **end if**
  **end for**
**end for**

The detailed algorithms are summarized in Appendix A.1 (Alg. 2 for on-policy and Alg. 3 for off-policy).

## 2.5 Experimental Evaluation

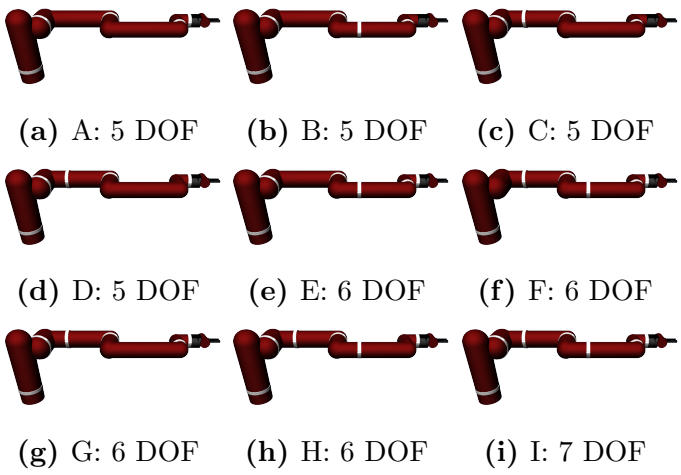
Our aim is to demonstrate the importance of conditioning the policy based on a hardware representation $v_h$ for transferring complicated policies between dissimilar robotic agents. We show performance gains on two diverse settings of manipulation and hopper.

### 2.5.1 Explicit Encoding

**Robot Hardwares:** We created a set of robot manipulators based on the Sawyer robot in MuJoCo [43]. The basic robot types are shown in Figure 2.2. By permuting the chronology of revolute joints and ensuring the robot design is feasible, we designed 9 types of robots (named as A, B,..., I) in which the first four are 5 DOF, the next four are 6 DOF, and the last one is 7 DOF, following the main feasible kinematic designs described in hardware design literature [44]. Each of these 9 robots were further varied with different link lengths and dynamics.

**Tasks:** We consider reacher and peg insertion tasks to demonstrate the effectiveness of explicit encoded representation. In reacher, robot starts from a random initial pose and it needs to move the end effector to the random target position. In peg-insertion, a peg is attached to the robot gripper and the task is to insert the peg into the hole on the table. It's considered a success only if the peg bottom goes inside the hole more than 0.03m. Goals are described by the 3D target positions $(x_g, y_g, z_g)$ of end effector (reacher) or peg bottom (peg insertion).

**States and Actions**: The states of both environments consist of the angles and velocities of all robot joints. Action is $n$-dimensional torque control over $n$ ($n \leq 7$) joints. Since we consider robots with different DOF in this paper, we use zero-padding for robots with $< 7$ joints to construct a fixed-length state vector for different robots. And the policy network always outputs 7-dimensional actions, but only the first $n$ elements are used as the control command.



**(a)** A: 5 DOF **(b)** B: 5 DOF **(c)** C: 5 DOF

**(d)** D: 5 DOF **(e)** E: 6 DOF **(f)** F: 6 DOF

**(g)** G: 6 DOF **(h)** H: 6 DOF **(i)** I: 7 DOF

**Figure 2.2:** Robots with different DOF and kinematics structures. The white rings represent joints. There are 4 variants of 5 and 6 DOF robots due to the different placements of joints.

**Robot Representation**: As mentioned in section 2.4.1, $v_h$ is explicitly constructed to represent the robot kinematic chain: $v_h = P_{-1} \oplus P_0 \oplus \cdots \oplus P_{n-1}$. We use zero-padding for robots with $< 7$ joints to construct a fixed-length representation vector $v_h$ for different robots.

**Rewards**: We use binary sparse reward setting because sparse reward is more realistic in robotics applications. And we use DPPG+HER as the backbone training algorithm. The agent only gets $+1$ reward if POI is within $\epsilon$ euclidean distance of the desired goal position. Otherwise, it gets $-1$ reward. We use $\epsilon = 0.02$m in all experiments. However, this kind of sparse reward setting encourages the agent to complete the task using as less time steps as possible to maximize the return in an

episode, which encourages the agent to apply maximum torques on all the joints so that the agent can move fast. This is referred to as bangbang control in control theory[45]. Hence, we added action penalty on the reward.

More experiment details are shown in Appendix A.2.

#### 2.5.1.1 Does HCP-E improve performance?

To show the importance of hardware information as input to the policy network, we experiment on learning robotic skills among robots with different dynamics (joint damping, friction, armature, link mass) and kinematics (link length, kinematic structure, DOF). The 9 basic robot types are listed in Figure 2.2. We performed several leave-one-out experiments (train on 8 robot types, leave 1 robot type untouched) on these robot types. The sampling ranges for link length and dynamics parameters are shown in Table A.1 in Appendix A.2.1.1. We compare our algorithm with vanilla DDPG+HER (trained with data pooled from all robots) to show the necessity of training a universal policy network conditioned on hardware characteristics. Figure 2.3 shows the learning curves[4] of training on robot types A-G and I. It clearly shows that our algorithm HCP-E outperforms the baseline. In fact, DDPG+HER without any hardware information is unable to learn a common policy across multiple robots as different robots will behave differently even if they execute the same action in the same state. More leave-one-out experiments are shown in Appendix A.3.4.

#### 2.5.1.2 Is HCP-E capable of zero-shot transfer to unseen robot kinematic structure?

We now perform testing in the leave-one-out experiments. Specifically, we can test the zero-shot transfer ability of policy network on new type of robots. Table 2.1 shows the quantitative statistics about testing performance on new robot types that are different from training robot types. Each data[5] in the table is obtained by running the model on 1000 unseen test robots (averaged over 10 trials, 100 robots per trial) of that robot type but with different link lengths and dynamics.

[4]The learning curves are averaged over 5 random seeds on 100 testing robots and shaded areas represent 1 standard deviation.

[5]The success rate is represented by the mean and standard deviation

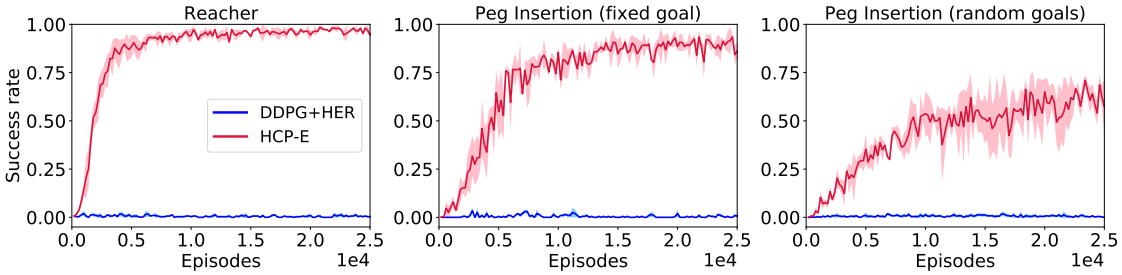



**Figure 2.3:** Learning curves for multi-DOF setup. Training robots contain Type A-G and Type I robots (four 5-DOF types, three 6-DOF types, one 7-DOF type). Each type has 140 variants with different dynamics and link lengths. The 100 testing robots used to generate the learning curves are from the same training robot types but with different link lengths and dynamics. (a): reacher task with random initial pose and target position. (b): peg insertion with fixed hole position. (c): peg insertion with hole position $(x, y, z)$ randomly sampled in a 0.2m box region. Notice that the converged success rate in (c) is only about 70%. This is because when we randomly generate the hole position, some robots cannot actually insert the peg into hole due to physical limit. Some hole positions are not inside the reachable space (workspace) of the robots. This is especially common in 5-DOF robots.

**Table 2.1:** Zero-shot testing performance on new robot type

| Exp. | Tasks | Training Robot Types | Testing Robot Type | Alg. | Success rate (%) |
|---|---|---|---|---|---|
| I | Reacher (random goals) | A-G + I | H | HCP-E | **92.50 ± 1.96** |
| II | | | | DDPG+HER | 0.20 ± 0.40 |
| III | | A-D + F-I | E | HCP-E | **88.00 ± 2.00** |
| IV | | | | DDPG+HER | 2.70 ± 2.19 |
| V | Peg Insertion (fixed goal) | A-G + I | H | HCP-E | **92.20 ± 2.75** |
| VI | | | | DDPG+HER | 0.00 ± 0.00 |
| VII | | A-D + F-I | E | HCP-E | **87.60 ± 2.01** |
| VIII | | | | DDPG+HER | 0.80 ± 0.60 |
| IX | | A-H | I | HCP-E | **65.60 ± 3.77** |
| X | | | | DDPG+HER | 0.10 ± 0.30 |
| XI | Peg Insertion (random goals) | A-G + I | H | HCP-E | **4.10 ± 1.50** |
| XII | | | | DDPG+HER | 0.10 ± 0.30 |
| XIII | | A-D + F-I | E | HCP-E | **76.10 ± 3.96** |
| XIV | | | | DDPG+HER | 0.00 ± 0.00 |
| XV | | A-H | I | HCP-E | **23.50 ± 4.22** |
| XVI | | | | DDPG+HER | 0.20 ± 0.40 |

From Table 2.1, it is clear that HCP-E still maintains high success rates when the policy is applied to new types of robots that have never been used in training, while DDPG+HER barely succeeds at controlling new types of robots at all. The difference between using robot types A-G+I and A-D+F-I (both have four 5-DOF types, three 6-DOF types, and one 7-DOF type) is that robot type H is harder for peg insertion task than robot type E due to its joint configuration (it removes joint $J_5$). As we can see from Exp. I and V, HCP-E got about 90% zero-shot transfer success rate even if it's applied on the hard robot type H. Exp. X and XVI show the model trained with only 5 DOF and 6 DOF being applied to 7-DOF robot type. We can see that it is able to get about 65% success rate in peg insertion task with fixed goal[6].

**Zero-shot transfer to a real Sawyer robot:** We show results on the multi-goal reacher task, as peg insertion required additional lab setup. Though the control frequency on the real robot is not as stable as that in simulation, we still found a high zero-shot transfer rate. For quantitative evaluation, we ran three policies on the real robot with results averaged over 20 random goal positions. **A** used the policy from Exp. I (HCP-E), **B** used the policy from Exp. II (DDPG+HER) while **C** used the policy trained with actual Sawyer CAD model in simulation with just randomized dynamics. The distance from target for the 20-trials are summarized in Figure 2.4. Despite of the large reality gap[7], HCP-E (BLUE) is able to reach the target positions with a high success rate (75%) [8]. DDPG+HER (RED) without hardware information was not even able to move the

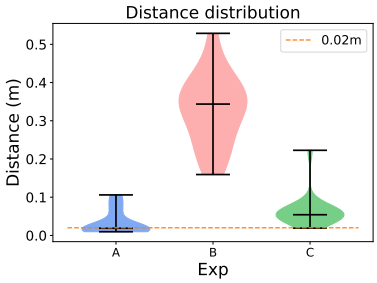
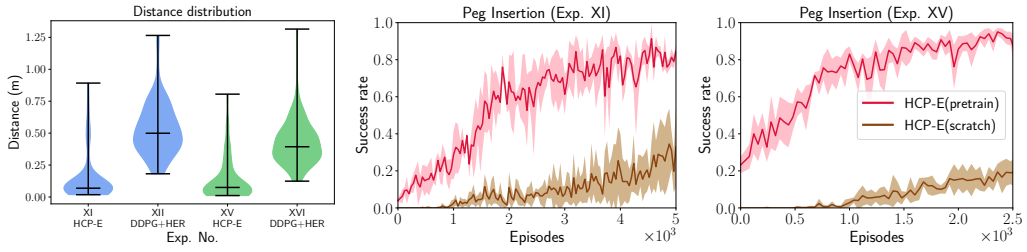



**Figure 2.4:** Testing distance distribution on a real sawyer robot. **A** used the policy from Exp. I, **B** used the policy from Exp. II, **C** used the policy trained with the actual Sawyer CAD model in simulation with randomized dynamics.

[6]Since we are using direct torque control without gravity compensation, the trivial solution of transferring where the network can regard the 7-DOF robot as a 6-DOF robot by keeping one joint fixed doesn't exist here.

[7]The reality gap is further exaggerated by the fact that we didn't do any form of gravity compensation in the simulation but the real-robot tests used the gravity compensation to make tests safer.

[8]The HCP-E policy resulted in a motion that was jerky on the real Sawyer robot to reach the target positions. This was because we used sparse reward during training. This could be mitigated with better reward design to enforce smoothness.




**(a)** **(b)** **(c)**

**Figure 2.5:** (a): Distribution (violin plots) of distance between the peg bottom at the end of episode and the desired position. The three horizontal lines in each violin plot stand for the lower extrema, median value, and the higher extrema. It clearly shows that HCP-E moves the pegs much closer to the hole than DDPG+HER. (b): The brown curve is the learning curve of training HCP-E on robot type H with different link lengths and dynamics in multi-goal setup from scratch. The pink curve is the learning curve of training HCP-E on same robots with pretrained model from Exp. XI. (c): Similar to (b), the training robots are robot type I (7 DOF) and the pretrained model is from Exp. XV. (b) and (c) show that applying the pretrained model that is trained on different robot types to a new robot type can accelerate the learning by a large margin.

arm close to the desired position.

**Fine-tuning the zero-shot policy:** Table 2.1 also shows that Exp. XI and Exp. XV have relatively low zero-shot success rates on new type of robots. Exp. XI is trained on easier 6-DOF robots (E, F, G) and applied to a harder 6-DOF robot type (H). Exp. XV is trained only on 5-DOF and 6-DOF robots and applied to 7-DOF robots (I). The hole positions are randomly sampled in both experiments. Even though the success rates are low, HCP-E is actually able to move the peg bottom close to the hole in most testing robots, while DDPG+HER is much worse, as shown in Figure 2.5a. We also fine-tune the model specifically on the new robot type for these two experiments, as shown in Figure 2.5b and Figure 2.5c. It's clear that even though zero-shot success rates are low, the model can quickly adapt to the new robot type with the pretrained model weights.

### 2.5.2 Implicit Encoding

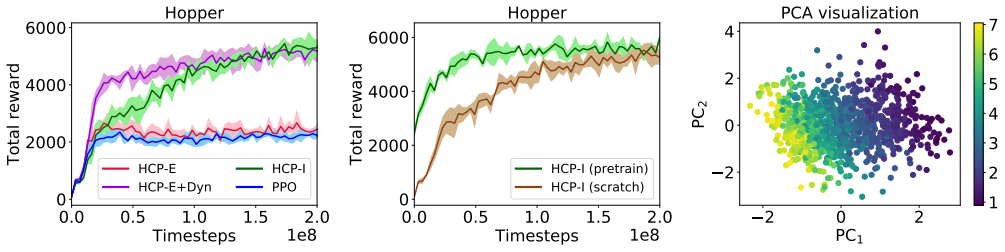
**Environment** HCP-E shows remarkable success on transferring manipulator tasks to different types of robots. However, in explicit encoding we only condition on the kinematics of the robot hardware. For unstable systems in robotics, such as

in legged locomotion where there is a lot of frequent nonlinear contacts [22], it is crucial to consider robot dynamics as well. We propose to learn an implicit, latent encoding (HCP-I) for each robot without actually using any kinematics and dynamics information. We evaluate the effectiveness of HCP-I on the 2D hopper [46]. Hopper is an ideal environment as it is an unstable system with sophisticated second-order dynamics involving contact with the ground. We will demonstrate that adding implicitly learned robot representation can lead to comparable performance to the case where we know the ground-truth kinematics and dynamics. To create robots with different kinematics and dynamics, we varied the length and mass of each hopper link, damping, friction, armature of each joint, which are shown in Table A.4 in Appendix A.2.

**Performance** We compare HCP-I with HCP-E, HCP-E+ground-truth dynamics, and vanilla PPO model without kinematics and dynamics information augmented to states. As shown in Figure 2.6a, HCP-I outperforms the baseline (PPO) by a large margin. In fact, with the robot representation $v_h$ being automatically learned, we see that HCP-I achieves comparable performance to HCP-E+Dyn that uses both kinematics and dynamics information, which means the robot representation $v_h$ learns the kinematics and dynamics implicitly. Since dynamics plays a key role in the hopper performance which can be seen from the performance difference between HCP-E and HCP-E+Dyn, the implicit encoding method obtains much higher return than the explicit encoding method. This is because the implicit encoding method can automatically learn a good robot hardware representation and include the kinematics and dynamics information, while the explicit encoding method can only encode the kinematics information as dynamics information is generally unavailable.

**Transfer Learning on New Agents** We now apply the learned HCP-I model as a pretrained model onto new robots. However, since $v_h$ for the new robot is unknown, we fine-tune the policy parameters and also estimate $v_h$. As shown in Figure 2.6b, HCP-I with pretrained weights learns much faster than HCP-I trained from scratch. While in the current version, we do not show explicit few-shot results, one can train a regression network to predict the trained $v_h$ based on the agent's limited interaction with environment.

**Embedding Smoothness** We found the implicit encoding $v_h$ to be a smooth embedding space over the dynamics. For example, we only vary one parameter (the




**(a)** **(b)** **(c)**

**Figure 2.6:** (a): Learning curves using 1000 hoppers with different kinematics and dynamics in training. HCP-I is able to automatically learn a good robot representation such that the learning performance can be on par with HCP-E+Dyn where we use the ground-truth kinematics and dynamics values. And HCP-I has a much better performance than vanilla PPO. (b): If we use the pretrained HCP-I model (only reuse the hidden layers) from (a) on 100 new hoppers, HCP-I with pretrained weights learns much faster than training from scratch. (c): Embedding visualization. The colorbar shows the hopper torso mass value. We can see that the embedding is smooth as the color transition is smooth.

torso mass) and plot the resultant embedding vectors. Notice that we reduce the dimension of $v_h$ to 2 since we only vary torso mass. Figure 2.6c shows a smooth transition over torso mass (the color bar represents torso mass value, 1000 hoppers with different torso mass), where robots with similar mass are clustered together.

## 2.6 Conclusion

We introduced a novel framework of *Hardware Conditioned Policies* for multi-robot transfer learning. To represent the hardware properties as a vector, we propose two methods depending on the task: explicit encoding (HCP-E) and implicit encoding (HCP-I). HCP-E works well when task policy does not heavily depend on agent dynamics. It has an obvious advantage that it is possible to transfer the policy network to new robots in a zero-shot fashion. Even when zero-shot transfer gives low success rate, we showed that HCP-E actually brings the agents very close to goals and is able to adapt to the new robots very quickly with finetuning. When the robot dynamics is so complicated that feeding dynamics information into policy network helps improve learning, the explicit encoding is not enough as it can only encode the kinematics information and dynamics information is usually challenging and

sophisticated to acquire. To deal with such cases, we propose an implicit encoding scheme (HCP-I) to learn the hardware embedding representation automatically via back-propagation. We showed that HCP-I, without using any kinematics and dynamics information, can achieve good performance on par with the model that utilized both ground truth kinematics and dynamics information.

# Chapter 3

# Exploration Policies for Navigation

## 3.1 Introduction

Imagine your first day at a new workplace. If you are like most people, the first task you set for yourself is to become familiar with the office so that the next day when you have to attend meetings and perform tasks, you can navigate efficiently and seamlessly. To achieve that goal, you explore your office without the task context of target locations you have to reach and build a generic understanding of space. This step of task-independent exploration is quite critical yet often ignored in current approaches for navigation.

When it comes to navigation, currently there are two paradigms: (a) geometric reconstruction and path-planning based approaches [47, 48, 49], and (b) learning-based approaches [7, 8, 9, 10]. SLAM-based approaches, first build a map and then use localization and path planning for navigation. In doing this, one interesting question that is often overlooked is: How does one build a map? How should we explore the environment to build this map? Current approaches either use a human operator to control the robot for building the map (*e.g.* [50]), or use heuristics such as frontier-based exploration [51]. On the other hand for approaches that use learning, most only learn policies for specific tasks [7, 8, 9], or assume environments have already been explored [10]. Moreover, in context of such learning based approaches, the question of exploration is not only important at test time, but also at train time. And once again, this question is largely ignored. Current approaches either use

sample inefficient random exploration or make impractical assumptions about full map availability for generating supervision from optimal trajectories.

Thus, a big bottleneck for both these navigation paradigms is an exploration policy: a task-agnostic policy that explores the environment to either build a map or sample trajectories for learning a navigation policy in a sample-efficient manner. But how do we learn this task-independent policy? What should be the reward for such a policy? First possible way is to not use learning and use heuristic based approaches [51]. However, there are four issues with non-learning based approaches: (a) these approaches are brittle and fail when there is noise in ego-estimation or localization; (b) they make strong assumptions about free-space/collisions and fail to generalize when navigation requires interactions such as opening doors *etc.*; (c) they fail to capture semantic priors that can reduce search-space significantly; and (d) they heavily rely on specialized sensors such as range scanners. Another possibility is to learn exploration policies on training environments. One way is to use reinforcement learning (RL) with intrinsic rewards. Examples of intrinsic rewards can be "curiosity" where prediction error is used as reward signal or "diversity" which discourages the agent from revisiting the same states. While this seems like an effective reward, such approaches are still sample inefficient due to blackbox reward functions that can't be differentiated to compute gradients effectively. So, what would be an effective way to learn exploration policies for navigation?

In this chapter, we propose an approach for learning policies for exploration for navigation. We explore this problem from multiple perspectives: (a) architectural design, (b) reward function design, and (c) reward optimization. Specifically, from perspective of reward function and optimization, we take the alternative paradigm and use supervision from human explorations in conjunction with intrinsic rewards. We notice that bootstrapping of learning from small amount of human supervision aids learning of semantics (*e.g.* doors are pathways). It also provides a good initialization for learning using intrinsic rewards. From the perspective of architecture design, we explore how to use 3D information and use it efficiently while doing exploration. We study proxy rewards that characterize coverage and demonstrate how this reward outperforms other rewards such as curiosity. Finally, we show how experience gathered from our learned exploration policies improves performance at down-stream navigation tasks.

## 3.2 Related Work

Our work on learning exploration policies for navigation in real world scenes is related to active SLAM in classical robotics, and intrinsic rewards based exploration in reinforcement learning. As we study the problem of navigation, we also draw upon recent efforts that use learning for this problem. We survey related efforts in these three directions.

**Navigation in Classical Robotics.** Classical approaches to navigation operate by building a map of the environment, localizing the agent in this map, and then planning paths to convey the agent to desired target locations. Consequently, the problems of mapping, localization and path-planning have been very thoroughly studied [47, 48, 49]. However, most of this research starts from a human-operated traversal of the environment, and falls under the purview of passive SLAM. Active SLAM, or how to automatically traverse a new environment for building spatial representations is much less studied. [52] present an excellent review of active SLAM literature, we summarize some key efforts here. Past works have formulated active SLAM as Partially Observable Markov Decision Processes (POMDPs) [53], or as choosing actions that reduce uncertainty in the estimated map [54]. While these formulations enable theoretical analysis, they crucially rely on sensors to build maps and localize. Thus, such approaches are highly susceptible to measurement noise. Additionally, such methods treat exploration purely as a geometry problem, and entirely ignore semantic cues for exploration such as doors.

**Learning for Navigation.** In order to leverage such semantic cues for navigation, recent works have formulated navigation as a learning problem [7, 8, 9, 10]. A number of design choices have been investigated. For example, these works have investigated different policy architectures for representing space: [7] use feed-forward networks, [9] use vanilla neural network memory, [8] use spatial memory and planning modules, and [10] use semi-parametric topological memory. Different training paradigms have also been explored: [8] learn to imitate behavior of an optimal expert, [9] and [7] use extrinsic reward based reinforcement learning, [55] learn an inverse dynamics model on the demonstrated trajectory way-points from the expert, while [10] use self-supervision. In our work here, we build on insights from these past works. Our policy architecture and reward definition use spatial memory to achieve long-horizon exploration, and we

imitate human exploration demonstrations to boot-strap policy learning. However, in crucial distinction, instead of studying goal-directed navigation (either in the form of going to a particular goal location, or object of interest), we study the problem of autonomous exploration of novel environments in a task-agnostic manner. In doing so, unlike past works, we do not assume access to human demonstrations in the given novel test environment like [10], nor do we assume availability of millions of samples of experience or reward signals in the novel test environment like [7] or [9]. Moreover, we do not depend on extrinsically defined reward signals for training. We derive them using on-board sensors. This makes our proposed approach amenable to real world deployment. Learning based approaches [56, 57] have also been used to learn low-level collision avoidance policies. While they do not use task-context, they learn to move towards open space, without the intent of exploring the whole environment. [58] uses a differentiable map structure to mimic the SLAM techniques. Such works are orthogonal to our effort on exploration as our exploration policy can benefit from their learned maps instead of only using reconstructed occupancy map.

**Exploration for Navigation.** A notable few past works share a similar line of thought, and investigate exploration in context of reinforcement learning [59, 60, 61, 62, 63, 64]. These works design intrinsic reward functions to capture novelty of states or state-action transitions. Exploration policies are then learned by optimizing these reward functions using reinforcement learning. Our work is most similar to these works. However, we focus on this problem in the specific context of navigation in complex and realistic 3D environments, and propose specialized policy architectures and intrinsic reward functions. We experimentally demonstrate that these specializations improve performance, and how learning based exploration techniques may not be too far from real world deployment. [65] use a related reward function (pixel reconstruction) to learn policies to look around (and subsequently solve tasks). However, they do it in context of 360° images, and their precise reward can't be estimated intrinsically. [66] generate smooth movement path for high-quality camera scan by using time-varying tensor fields. [67] propose an information-theoretic exploration method using Gaussian Process regression and show experiments on simplistic map environments. [68] assume access to the ground-truth map and learn an optimized trajectory that maximizes the accuracy of the SLAM-derived map. In contrast, our learning policy directly tells the action that the agent should take next and estimates the map on

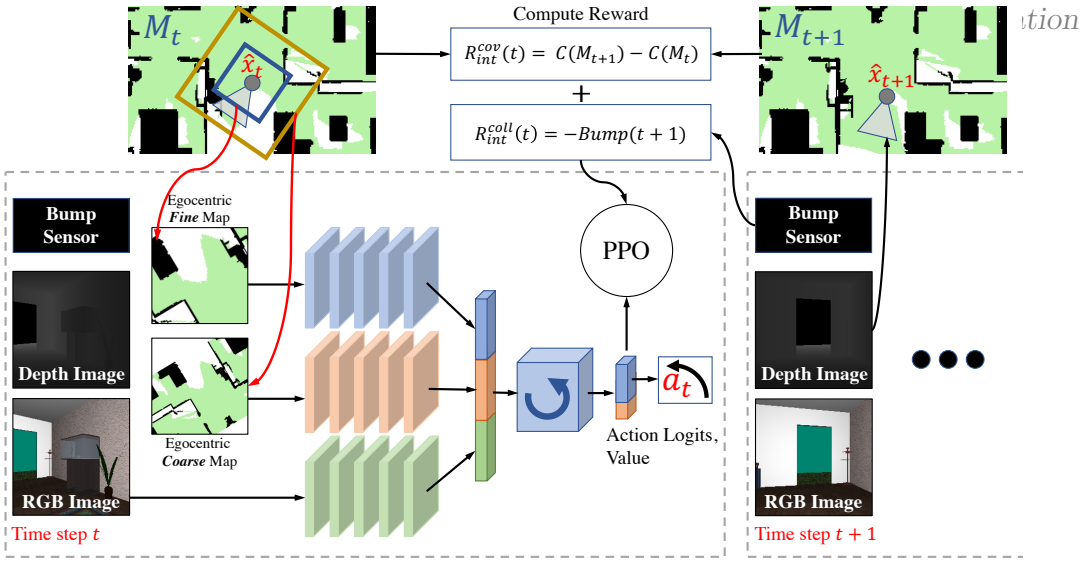



**Figure 3.1: Policy and Training Architecture:** Our scheme for learning exploration policies for navigation. We assume a mobile robot with an RGB-D camera and a bump sensor. We maintain a running estimate of its position $\hat{x}_t$. These position estimates are used to stitch together an approximate map of the environment using the depth images. The exploration policy $\pi_e$ is a recurrent network takes egocentric crops of this map at two scales, and the current RGB image as input. The policy is trained using an intrinsic coverage reward that is computed from the approximate map and a collision penalty obtained from the bump sensor.

the fly.

**System Identification.** Finally, the general idea of exploration prior to goal-driven behavior, is also related to the classical idea of system identification [69]. Recent interest in this idea with end-to-end learning [40, 70, 71] has been shown to successfully adapt to novel mazes. In comparison, we tackle navigation in complex and realistic 3D environments, with very long spatio-temporal dependencies.

## 3.3 Approach

Let us consider a mobile agent that is equipped with an RGB-D camera and that can execute basic movement macro-actions. This agent has been dropped into a novel environment that it has never been in before. We want to learn an exploration policy $\pi_e$ that enables this agent to efficiently explore this new environment. A successful exploration of the new environment should enable the agent to accomplish down-stream tasks in this new environment efficiently.

We formulate estimation of $\pi_e$ as a learning problem. We design a reward function

that is estimated entirely using on-board sensors, and learn $\pi_e$ using RL. Crucially, $\pi_e$ is learned on a set of environments $\mathcal{E}_{train}$, and tested on a held-out set of environments $\mathcal{E}_{test}$, *i.e.*, $\mathcal{E}_{train} \bigcap \mathcal{E}_{test} = \phi$.

The key novelties in our work are: a) the design of the policy $\pi_e$, b) the design of the reward function $R$, and c) use of human demonstrations to speed up training of $\pi_e$. The design of the policy and the reward function depend on a rudimentary map of the environment that we maintain over time. We first describe the map construction procedure, and then detail the aforementioned aspects.

### 3.3.1 Map Construction

As the agent moves around, it uses its depth camera to build and update a map of the world around it. This is done by maintaining an estimate of the agent's location over time and projecting 3D points observed in the depth image into an allocentric map of the environment.

More specifically, let us assume that the agent's estimate of its current location at time step $t$ is $\hat{x}_t$, and that the agent starts from origin (*i.e.* $\hat{x}_0 = 0$). When the agent executes an action $a_t$ at time step $t$, it updates its position estimate through a known transition function $f$, *i.e.*, $\hat{x}_{t+1} = f(\hat{x}_t, a_t)$.

The depth image observation at time step $t+1$, $D_{t+1}$ is back-projected into 3D points using known camera intrinsic parameters. These 3D points are transformed using the estimate $\hat{x}_{t+1}$, and projected down to generate a 2D map of the environment (that tracks what parts of space are known to be traversable or known to be non-traversable or unknown). Note that, because of slippage in wheels *etc.*, $\hat{x}_{t+1}$ may not be the same as the true relative location of the agent from start of episode $x_{t+1}$. This leads to aliasing in the generated map. We do not use expensive non-linear optimization (bundle adjustment) to improve our estimate $\hat{x}_{t+1}$, but instead rely on learning to provide robustness against this mis-alignment. We use the new back-projected 3D points to update the current map $M_t$ and to obtain an updated map $M_{t+1}$.

### 3.3.2 Policy Architecture

Before we describe our policy architecture, let's define what are the features that a good exploration policy needs: (a) good exploration of a new environment requires an agent to meaningfully move around by detecting and avoiding obstacles; (b) good policy also requires the agent to identify semantic cues such as doors that may facilitate exploration; (c) finally, it requires the agent to keep track of what parts of the environment have or have not been explored, and to estimate how to get to parts of the environment that may not have been explored.

This motivates our policy architecture. We fuse information from RGB image observations and occupancy grid-based maps. Information from the RGB image allows recognition of useful semantic cues. While information from the occupancy maps allows the agent to keep track of parts of the environment that have or have not been explored and to plan paths to unexplored regions without bumping into obstacles.

The policy architecture is shown in Fig. 3.1. We describe it in detail here:

1. **Information from RGB images**: RGB images are processed through a CNN. We use a ResNet-18 CNN that has been pre-trained on the ImageNet classification task, and can identify semantic concepts in images.
2. **Information from Occupancy Maps**: We derive an occupancy map from past observations (as described in Sec. 3.3.1), and use it with a ResNet-18 CNN to extract features. To simplify learning, we do not use the allocentric map, but transform it into an egocentric map using the estimated position $\hat{x}_t$ (such that the agent is always at the center of the map, facing upwards). This map canonicalization aids learning. It allows the CNN to not only detect unexplored space but to also locate it with respect to its current location. Additionally, we use two such egocentric maps, a coarse map that captures information about a $40m \times 40m$ area around the agent, and a detailed map that describes a $4m \times 4m$ neighborhood. Some of these design choices were inspired by recent work from [8], [72] and [73], though we make some simplifications.
3. **Recurrent Policy**: Information from the RGB image and maps is fused and passed into an RNN. Recurrence can allow the agent to exhibit coherent behavior.

### 3.3.3 Coverage Reward

We now describe the intrinsic reward function that we use to train our exploration policy. We derive this intrinsic rewards from the map $M_t$, by computing the coverage. Coverage $C(M_t)$ is defined as the total area in the map that is known to be traversable or known to be non-traversable. Reward $R^{cov}_{int}(t)$ at time step $t$ is obtained via gain in coverage in the map: $C(M_{t+1}) - C(M_t)$. Intuitively, if the current observation adds no obstacles or free-space to the map then it adds no information and hence no reward is given. We also use a collision penalty, that is estimated using the bump sensor, $R^{coll}_{int}(t) = -Bump(t+1)$, where $Bump(t+1)$ denotes if a collision occurred while executing action $a_t$. $R^{cov}_{int}$ and $R^{coll}_{int}$ are combined to obtain the total reward.

### 3.3.4 Training Procedure

Finally, we describe how we optimize the policy. Navigating in complex realistic 3D houses, requires long-term coherent behavior over multiple time steps, such as exiting a room, going through doors, going down hallways. Such long-term behavior is hard to learn using reinforcement learning, given sparsity in reward. This leads to excessively large sample complexity for learning. To overcome this large sample complexity, we pre-train our policy to imitate human demonstrations of how to explore a new environment. We do this using trajectories collected from AMT workers as they were answering questions in House3D [74]. We ignore the question that was posed to the human, and treat these trajectories as a proxy for how a human will explore a previosuly unseen environment. After this phase of imitation learning, $\pi_e$ is further trained via policy gradients [75] using proximal policy optimization (PPO) from [76].

## 3.4 Experiments

The goal of this paper is to build agents that can autonomously explore novel complex 3D environments. We want to understand this in context of the different choices we made in our design, as well as how our design compares to alternate existing techniques for exploration. While coverage of the novel environment is a good task-agnostic metric, we also design experiments to additionally quantify the utility of generic

task-agnostic exploration for downstream tasks of interest. We first describe our experimental setup that consists of complex realistic 3D environments and emphasizes the study of generalization to novel environments. We then describe experiments that measure task-agnostic exploration via coverage. And finally, we present experiments where we use different exploration schemes for downstream navigation tasks.

### 3.4.1 Experimental Setup

We conducted our experiments on the House3D simulation environment [77]. House 3D is based on realistic apartment layouts from the SUNCG dataset [78] and simulates first-person observations and actions of a robotic agent embodied in these apartments. We use 20 houses each for training and testing. These sets are sampled from the respective sets in House 3D, and do not overlap. That is, testing is done on a set of houses not seen during training. This allows us to study generalization, *i.e.*, how well our learned policies perform in novel, previously unseen houses. We made one customization to the House 3D environment: by default doors in House 3D are rendered but not used for collision checking. We modified House 3D to also not render doors, in addition to not using them for collision checking.

**Observation Space.** We assume that the agent has an RGB-D camera with a field of view of 60°, and a bump sensor. The RGB-D camera returns a regular RGB image and a depth image that records the distance (depth information is clipped at 3m) of each pixel from the camera. The bump sensor returns if a collision happened while executing the previous action.

**Action Space.** We followed the same action space as in EmbodiedQA [74], with 6 motion primitives: *move forward* 0.25m, *move backward* 0.25m, *strafe left* 0.25m, *strafe right* 0.25m, *turn left* 9°, and *turn right* 9°.

**Extrinsic Environmental Reward.** We do not use any externally specified reward signals from the environment: $R_{ext}(t) = 0$.

**Intrinsic Reward.** As described in Sec. 3.3.3, the intrinsic reward of the agent is based upon the map that it constructs as it moves around (as described in Sec. 3.3.1), and readings from the bump sensor. Reward $R_{int}(t) = \alpha R^{\text{cov}}_{int}(t) + \beta R^{\text{coll}}_{int}(t)$. Here $R^{\text{cov}}_{int}$ is the coverage reward, and $R^{\text{coll}}_{int}$ is the collision reward. $\alpha, \beta$ are hyper-parameters to trade-off how aggressive the agent is.

**Training.** As described in Sec. 3.3.4, we train policies using imitation of human trajectories and RL.

1. *Imitation from Human Exploration Trajectories.* We leverage human exploration trajectories collected from AMT workers as they answered questions for the EmbodiedQA task [74]. We ignore the question that was posed to the AMT worker, and train our policy to simply mimic the actions that the humans took. As the humans were trying to answer these questions in previously unseen environments, we assume that these trajectories largely exhibit exploration behavior. We used a total of 693 human trajectories for this imitation.
2. *Reinforcement Learning*: After learning via imitation, we further train the policy using reinforcement learning on the training houses. We use PPO [76] to optimize the intrinsic reward defined above. At the start of each episode, the agent is initialized at a random location inside one of the training houses. Each episode is run for 500 time-steps. We run a total of 6400 episodes which amounts to a total of 3.2M steps of experience.

**Baselines.** We next describe the various baseline methods that we experimented with. We implemented a classical baseline that purely reasons using geometry, and a learning baseline that uses curiosity for exploration.

1. *Frontier-based Exploration.* As a classical baseline, we experimented with frontier-based exploration [51, 79]. This is a purely geometric method that utilizes the built map $M_t$. Every iteration, it samples a point in currently unexplored space, and plans a path towards it from the current location (unobserved space is assumed to be free). As the plan is executed, both the map and the plan are updated. Once the chosen point is reached, this process is repeated.
2. *Curiosity-based Exploration.* The next baseline we tried was curiosity-based exploration. In particular, we use the version proposed by [61] that uses prediction error of a forward model as reward. We use the modifications proposed by [80], and only train a forward model. We prevent degeneracy in forward model by learning it in the fixed feature space of a ResNet-18 model that has been pre-trained on the ImageNet classification task.

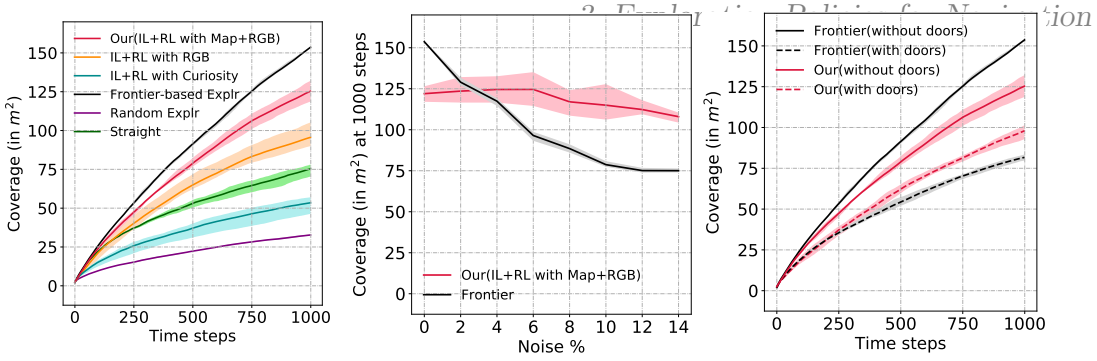



**Figure 3.2: Coverage Performance.** Policies are tested on 100 exploration runs (5 random start locations each on 20 testing houses). **Left figure** plots average coverage as a function of number of time steps in the episode. **Center figure** studies impact of noise in state estimation. We plot average coverage at end of episode as a function of amount of noise. **Right figure** studies the case when there is a mis-match between geometry and affordance, and plots average coverage as a function of time steps in episode. All plots report mean performance over 3 runs, and the shaded area represents the minimum and maximum performance.

### 3.4.2 Coverage Quality

We first measure exploration by itself, by measuring the *true* coverage of the agent. We compute the true coverage using the map as described in Sec. 3.3.1, except for using the true location of the agent rather than the estimated location (*i.e.* $x_t$ instead of $\hat{x}_t$). We study the following three scenarios:

**Without Estimation Noise:** We first evaluate the performance of the agent that is trained and tested without observation noise, that is, $\hat{x}_t = x_t$. Note that *this setting is not very realistic as there is always observation error in an agent's estimate of its location.* It is also highly favorable to the frontier-based exploration agent, which very heavily relies on the accuracy of its maps. Fig. 3.2(left) presents the performance of different policies for this scenario. We first note that the curiosity based agent (**IL + RL with Curiosity**) explores better than a random exploration policy (that executes a random action at each time step). **Straight** is a baseline where the agent moves along a straight line and executes a random number of 9° turns when a collision occurs, which is a strategy used by many robot vacuums. Such strategy does not require RGB or depth information, and performs better than the curiosity based agent. However, both policies are worse than an RGB only version of our method, an RGB only policy that is trained with our coverage reward (**IL + RL with RGB**). Our full system (**IL + RL with Map + RGB**) that also uses

maps as input performs even better. Frontier-based exploration (**Frontier-based Explr**) has the best performance in this scenario. As noted, this is to expect as this method gets access to perfect, fully-registered maps, and employs optimal path planning algorithm to move from one place to another. It is also worth noting that, it is hard to use such classical techniques in situations where we only have a RGB images. In contrast, learning allows us to easily arrive at policies that can explore using only RGB images at test time.

**With Estimation Noise:** We next describe a more realistic scenario that has estimation noise, *i.e.* $\hat{x}_{t+1}$ is estimated using a noisy dynamics function. In particular, we add truncated Gaussian noise to the transition function $f$ at each time step. The details of the noise generation is elaborated in Appendix B.3.4. The noise compounds over time. Even though such a noise model leads to compounding errors over time (as in the case of a real robot), we acknowledge that this simple noise model may not perfectly match noise in the real world. Fig. 3.2(center) presents the coverage area at the end of episode (1000 time steps) as a function of the amount of noise introduced[1]. When the system suffers from sensor noise, the performance of the frontier-based exploration method drops rapidly. In contrast, our learning-based agent that wasn't even trained with any noise continues to performs well. Even at relatively modest noise of 4% our learning based method already does better than the frontier-based agent. We additionally note that our agent can even be trained when the intrinsic reward estimation itself suffers from state estimation noise: for instance performance with 10% estimation noise (for intrinsic reward computation and map construction) is $98.9m^2$, only a minor degradation from $117.4m^2$ (10% estimation noise for map construction at test time only).

**Geometry and Affordance Mismatch:** Next, to emphasize the utility of learning for this task, we experiment with a scenario where we explicitly create a mismatch between geometry and affordance of the environment. We do this by rendering doors, but not using them for collision checking (*i.e.* the default House 3D environment). This setting helps us investigate if learning based techniques go beyond simple geometric reasoning in any way. Fig. 3.2(right) presents performance

[1]A noise of $\eta$ means we sample perturbations from a truncated Gaussian distribution with zero-mean, $\eta$ standard deviation, and a total width of $2\eta$. This sampled perturbation is scaled by the step length (25cm for $x, y$ and 9° for azimuth $\theta$) and added to the state at each time step.

curves. We see that there is a large drop in performance for the frontier-based agent. This is because it is not able to distinguish doors from other obstacles, leading to path planning failures. However, there is a relatively minor drop in performance of our learning-based agent. This is because it can learn about doors (how to identify them in RGB images and the fact that they can be traversed) from human demonstrations and experience during reinforcement learning.

#### 3.4.2.1 Ablation Study

We also conducted ablations of our method to identify what parts of our proposed technique contribute to the performance. We do these ablations in the setting without any estimation noise, and use coverage as the metric.

**Imitation Learning:** We check if pre-training with imitation learning is useful for this task. We test this by comparing to the models that were trained only with RL using the coverage reward. The left two plots in Fig. 3.3 shows performance of agents with following combinations: *RL*: policy trained with PPO only, *IL*: policy trained with imitation learning, *Map*: policy uses constructed maps as input, *RGB*: policy uses RGB images as input. In all settings, pre-training with imitation learning helps improve performance, though training with RL improves coverage further. Also, policies trained using RL have a fairly large variance. Imitation learning also helps reduce the variance.

**RGB Observations and Map:** Fig. 3.3 (left) and Fig. 3.3 (center) respectively show that both RGB images and map inputs improve performance.

**Intrinsic Reward:** We also compare our intrinsic reward design with extrinsic reward design, as shown in Fig. 3.3 (right). The extrinsic reward is setup as follows: we randomly generated a number of locations evenly distributed across the traversable area of the houses, where the agent will get a positive reward if the agent is close to any of these locations. Once the agent gets a reward from a location, this location will be no longer taken into account for future reward calculation. We tried two settings where we place 1 or 4 reward-yielding objects per $m^2$. We can see that our coverage map reward provides a better reward signal and in fact can be estimated intrinsically without needing to instrument the environment.

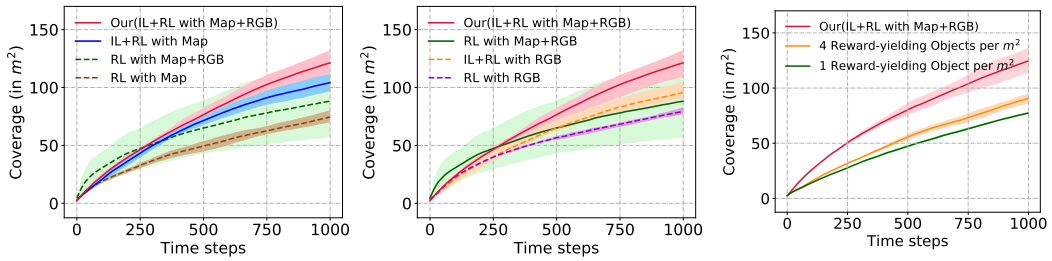


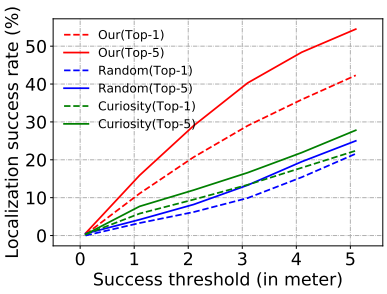

**Figure 3.3: Ablation Study:** We report average coverage as a function of time step in episode. As before we plot mean over 3 runs and minimum and maximum performance. **Left figure** shows that using the RGB image helps improve performance, **center figure** shows that using the map helps improve performance. We can also see that imitation learning improves coverage and reduces the variance in performance. **Right figure** shows the comparison between different reward design. We can see that our intrinsic reward enables the agent to explore more efficiently in the testing time.

### 3.4.3 Using Exploration for Downstream Task

Now that we have established how to explore well, we next ask if task-agnostic exploration is even useful for downstream tasks. We show this in context of goal-driven navigation. We execute the learned exploration policy $\pi_e$ (for 1500 time steps) to explore a new house, and collect experience (set of images $\mathcal{I}$ along with their pose $\mathbf{p}$). Once this exploration has finished, the agent is given different navigation tasks. These navigation tasks put the agent at an arbitrary location in the environment, and give it a *goal image* that it needs to reach. More specifically, we reset the agent to 50 random poses (positions and orientations) in each testing house (20 testing houses in total) and get the RGB camera view. The agent then needs to use the experience of the environment acquired during exploration to efficiently navigate to the desired target location. The efficiency of navigation on these test queries measures the utility of exploration.

This evaluation requires a navigation policy $\pi_n$, that uses the exploration experience and the goal image to output actions that can convey the agent to the desired target location. We opt for a simple policy $\pi_n$. $\pi_n$ first localizes the target image using nearest neighbor matching to the set of collected images $\mathcal{I}$ (in ImageNet pre-trained ResNet-18 feature space). It then plans a path to the this estimated target location using a occupancy map computed from $\mathcal{I}$. We do this experiment in the setting without any state estimation noise.

We independently measure the effectiveness of exploration data for a) localization of the given target images, and b) path planning efficiency in reaching desired locations. **Localization performance:** We measure the distance between the agent's estimate of the goal image location, and the true goal image location. Fig. 3.4 (top) plots the success at localization as a function of the success threshold (distance at which we consider a localization as correct). We report top-1 and top-5 success rates. We compare to the random exploration baseline and curiosity-driven baseline which serve to measure the hardness of the task. We see our exploration scheme performs well. **Path planning efficiency:** Next, we measure how efficiently desired goal locations can be reached. We measure performance using the SPL metric as described by [81] (described in the appendix, higher is better). We compare against a baseline that does not have any prior experience in this environment and derives all map information on the fly from goal driven behavior (going to the desired test location). Both agents take actions based on the shortest-path motion planning algorithm. Once again these serve as a measure of the hardness of the topology of the underlying environment. As shown in Fig. 3.4 (bottom), using exploration data from our policy improves efficiency of paths to reach target locations.




| SPL (No exploration) | SPL (Our exploration) |
|---|---|
| 0.69 | **0.87** |

**Figure 3.4: Exploration for downstream tasks.** We evaluate utility of exploration for the down-stream navigation tasks. **Top plots** shows effectiveness of the exploration trajectory for localization of goal images in a new environment. **Bottom table** compares efficiency of reaching goals without and with maps built during exploration.

## 3.5 Discussion

In this chapter, we motivated the need for learning explorations policies for navigation in novel 3D environments. We showed how to design and train such policies, and how experience gathered from such policies enables better performance for downstream tasks. We think we have just scratched the surface of this problem, and hope this inspires future research towards semantic exploration using more expressive policy architectures, reward functions and training techniques.

# Chapter 4

# Conclusions

This thesis explores how to improve the generalization capability of deep reinforcement learning algorithms using prior knowledge of the tasks. For multi-robot transfer learning problem, we found that augmenting the state vector with a vector that can represent the hardware characteristics enables the policy to generalize over robots with different kinematic structures, link lengths, and dynamics. Such hardware vectors can be either explicitly constructed given prior knowledge of the agents or implicitly learned via back-propagation without using any prior knowledge. As for learning exploration policies for navigation, a policy with projected 3D dense map (occupancy map) as an augmented state input can be trained efficiently to retain its ability to explore when placed in new environments. Our experiments show that such policies significantly outperform policies with only RGB images as the input. The key insight here is that the projected 3D dense map can serve as a spatial memory, which resolves the forgetting issues that are typical in RNNs. Such long-horizon exploration tasks require strong memory of the states that have been visited to make the exploration efficient. And the spatial memory performs much better than LSTMs or GRUs in such tasks. In conclusion, using prior knowledge can significantly improve the learning speed as well as the generalization capabilities of deep reinforcement learning algorithms, even in model-free algorithms.

# Appendix A

# Multi-Robot Transfer Learning

## A.1 Algorithms

### A.1.1 Algorithms

In this section, we present two detailed practical algorithms based on the HCP concept. Alg. 2 is HCP based on PPO which can be used to solve tasks with dense reward. Alg. 3 is HCP based on DDPG+HER which can be used to solve multi-goal tasks with sparse reward.

**Algorithm 2** Hardware Conditioned Policy (HCP) - on-policy

Initialize PPO algorithm
Initialize a robot pool $\mathcal{P}$ of size $N$ with robots in different dynamics and kinematics
**for** episode = 1, M **do**
  **for** actor=1, K **do**
    Sample a robot instance $\mathcal{I} \in \mathcal{P}$
    Sample an initial state $s_0$
    Retrieve the robot hardware representation vector $v_h$
    Augment $s_0$:
      $\hat{s}_0 \leftarrow s_0 \oplus v_h$
    **for** t = 0, T-1 **do**
      Sample action $a_t \leftarrow \pi(\hat{s}_t)$ using current policy
      Execute action $a_t$, receive reward $r_t$, observe new state $s_{t+1}$, and augmented state $\hat{s}_{t+1}$
    **end for**
    Compute advantage estimates $A_0, A_1, ..., A_{T-1}$
  **end for**
  **for** n=1,W **do**
    Optimize actor and critic networks with PPO via minibatch gradient descent
    **if** $v_h$ is to be learned **then**
      update $v_h$ via gradient descent in the optimization step as well
    **end if**
  **end for**
**end for**

**Algorithm 3** Hardware Conditioned Policy (HCP) - off-policy

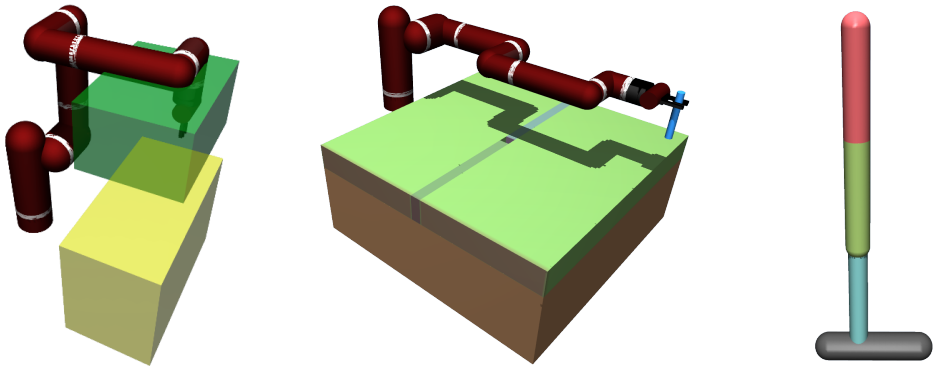
Initialize DDPG algorithm
Initialize experience replay buffer $\mathcal{R}$
Initialize a robot pool $\mathcal{P}$ of size $N$ with robots in different dynamics and kinematics
**for** episode = 1, M **do**
  Sample a robot instance $\mathcal{I} \in \mathcal{P}$
  Sample a goal position $g$ and an initial state $s_0$
  Retrieve the robot hardware representation vector $v_h$
  Augment $s_0$:
    $\hat{s}_0 \leftarrow s_0 \oplus g \oplus v_h$
  **for** t = 0, T-1 **do**
    Sample action $a_t \leftarrow \pi_b(\hat{s}_t)$ using behavioral policy
    Execute action $a_t$, receive reward $r_t$, observe new state $s_{t+1}$, and augmented state $\hat{s}_{t+1}$
    Store $(\hat{s}_t, a_t, r_t, \hat{s}_{t+1})$ into $\mathcal{R}$
  **end for**
  Augment $\mathcal{R}$ with pseudo-goals via HER
  **for** n=1,W **do**
    Optimize actor and critic networks with DDPG via minibatch gradient descent
    **if** $v_h$ is to be learned **then**
      update $v_h$ via gradient descent in the optimization step as well
    **end if**
  **end for**
**end for**

## A.2 Experiment Details

We performed experiments on three environments in this paper: reacher, peg insertion, and hopper, as shown in Figure A.1. Videos of experiments are available at: https://sites.google.com/view/robot-transfer-hcp.

### A.2.1 Reacher and Peg Insertion

The reason why we choose reacher and peg insertion task is that most of manipulator tasks like welding, assembling, grasping can be seen as a sequence of reacher tasks in essence. Reacher task is the building block of many manipulator tasks. And peg insertion task can further show the control accuracy and robustness of the policy network in transferring torque control to new robots.



**(a)** reacher **(b)** peg insertion **(c)** hopper

**Figure A.1:** (a): reacher, the green box represents end effector initial position distribution, and the yellow box represents end effector target position distribution. (b): peg insertion. The white rings in (a) and (b) represent joints. (c): hopper.

#### A.2.1.1 Robot Variants

During training time, we consider 9 basic robot types (named as Type A,B,...,I) as shown in Figure 2.2 which have different DOF and joint placements. The 5-DOF and 6-DOF robots are created by removing joints from the 7-DOF robot.

We also show the length range of each link and dynamics parameter ranges in Table A.1. The link name and joint name conventions are defined in Figure A.2.

Notice that damping values ranged from $[0, 1), (1, +\infty)$ are called underdamped and overdamped systems respectively. As these systems have very different dynamics characteristics, 50% of the damping values sampled are less than 1, and the rest 50% are greater than or equal to 1.

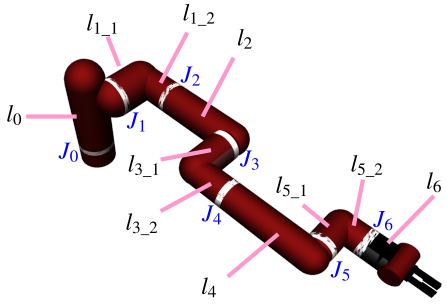



**Figure A.2:** Link name and joint name convention

**Table A.1:** Manipulator Parameters

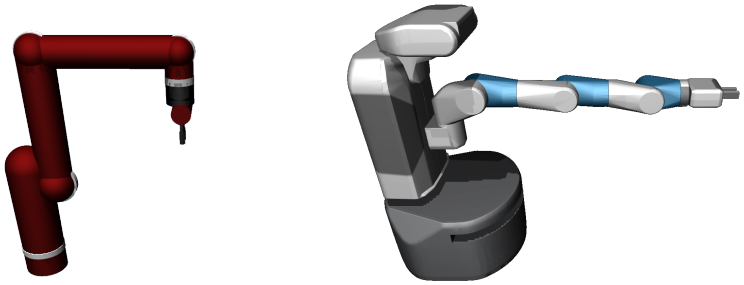
| Kinematics | |
|---|---|
| Links | Length Range (m) |
| $l_0$ | $0.290 \pm 0.10$ |
| $l_{1\_1}$ | $0.119 \pm 0.05$ |
| $l_{1\_2}$ | $0.140 \pm 0.07$ |
| $l_2$ | $0.263 \pm 0.12$ |
| $l_{3\_1}$ | $0.120 \pm 0.06$ |
| $l_{3\_2}$ | $0.127 \pm 0.06$ |
| $l_4$ | $0.275 \pm 0.12$ |
| $l_{5\_1}$ | $0.096 \pm 0.04$ |
| $l_{5\_2}$ | $0.076 \pm 0.03$ |
| $l_6$ | $0.049 \pm 0.02$ |
| Dynamics | |
| damping | $[0.01, 30]$ |
| friction | $[0, 10]$ |
| armature | $[0.01, 4]$ |
| link mass | $[0.25, 4]\times$ default mass |

Even though we only train with robot types listed in Figure 2.2, our policy can be directly transferred to other new robots like the Fetch robot shown in Figure A.3.

#### A.2.1.2 Hyperparameters

We closely followed the settings in original DDPG paper. Actions were added at the second hidden layer of $Q$. All hidden layers used scaled exponential linear unit (SELU) as the activation function and we used Adam optimizer. Other hyperparameters are summarized in Table A.2.

**Initial position distributions**: For reacher task, the initial position of end effector is randomly sampled from a box region $0.3\text{m} \times 0.4\text{m} \times 0.2\text{m}$. For peg insertion task, all robots start from a horizontal fully-expanded pose.



**(a)** J: new 5-DOF robot

**(b)** K: new 7-DOF robot (Fetch)

**Figure A.3:** New types of robot. We used the model trained in Exp. V and directly applied to robot shown in (a) and Fetch robot shown in (b). We tested the model on 1000 unseen test robots (averaged over 10 trials, 100 robots per trial) of type J (as shown in (a)), and got $86.30 \pm 4.41\%$ success rate. We tested on the fetch robot in (b) 10 times and got 100% success rate.

**Goal distributions**: For reacher task, the target end effector position region is a box region 0.3m $\times$ 0.6m $\times$ 0.4m which is located 0.2m below the initial position sampling region. For peg insertion task, we have experiments on hole position fixed and hole position randomly moved. If the hole position is to be randomly moved, the table's position will be randomly sampled from a box region 0.2m $\times$ 0.2m $\times$ 0.2m.

**Rewards**: As mentioned in paper, we add action penalty on rewards so as to avoid bang-bang control. The reward is defined as: $r(s_t, a_t, g) =_{\pm} (\epsilon - \|s_{t+1}(\text{POI}) - g\|_2) - \beta \|a_t\|_2^2$, where $s_{t+1}(\text{POI})$ is the position of the point of interest (POI, end effector in reacher and peg bottom in peg insertion) after the execution of action $a_t$ in the state $s_t$, $\beta$ is a hyperparameter $\beta > 0$ and $\beta \|a_t\|_2^2 \ll 1$.

**Success criterion**: For reacher task, the end effector has to be within 0.02m Euclidean distance to the target position to be considered as a success. For peg insertion task, the peg bottom has to be within 0.02m Euclidean distance to the target peg bottom position to be considered as a success. Since the target peg bottom position is always 0.05m below the table surface no matter how table moves, so the peg has to be inserted into the hole more than 0.03m.

**Observation noise**: We add uniformly distributed observation noise on states (joint angles and joint velocities). The noise is uniformly sampled from $[-0.02, 0.02]$ for both joint angles (rad) and joint velocities ($\text{rad}\,\text{s}^{-1}$).

**Table A.2:** Hyperparameters for reacher and peg insertion tasks

| | |
|---|---|
| number of training robots for each type | 140 |
| success distance threshold $\epsilon$ | 0.02m |
| maximum episode time steps | 200 |
| actor learning rate | 0.0001 |
| critic learning rate | 0.0001 |
| critic network weight decay | 0.001 |
| hidden layers | 128-256-256 |
| discount factor $\gamma$ | 0.99 |
| batch size | 128 |
| warmup episodes | 50 |
| experience replay buffer size | 1000000 |
| network training iterations after each episode | 100 |
| soft target update $\tau$ | 0.01 |
| number of future goals $k$ | 4 |
| action penalty coefficient $\beta$ | 0.1 |
| robot control frequency | 50Hz |

**Table A.3:** Hyperparameters for hopper

| | |
|---|---|
| number of training hoppers | 1000 |
| number of actors K | 8 |
| maximum episode time steps | 2048 |
| learning rate | 0.0001 |
| hidden layers | 128-128 |
| discount factor $\gamma$ | 0.99 |
| GAE parameter $\lambda$ | 0.95 |
| clip ratio $\eta$ | 0.2 |
| batch size | 512 |
| $v_h$ dimension | 32 |
| network training epochs after each rollout | 5 |
| value function loss coefficient $c_1$ | 0.5 |
| entropy loss coefficient $c_2$ | 0.015 |

### A.2.2 Hopper

We used the same reward design as the hopper environment in OpenAI Gym. As it's a dense reward setting, we use PPO for this task. All hidden layers used scaled exponential linear unit (SELU) as the activation function and we used Adam optimizer. Other hyperparameters are summarized in Table A.3. And the sampling ranges of link lengths and dynamics are shown in Table A.4.

**Table A.4:** Hopper Parameters

| Kinematics | | |
|---|---|---|
| Links | Length Range (m) | |
| torso | $0.40 \pm 0.10$ | |
| thigh | $0.45 \pm 0.10$ | |
| leg | $0.50 \pm 0.15$ | |
| foot | $0.39 \pm 0.10$ | |
| Dynamics | | |
| damping | $[0.01, 5]$ | |
| friction | $[0, 2]$ | |
| armature | $[0.1, 2]$ | |
| link mass | $[0.25, 2]\times$ mass | default |

## A.3 Supplementary Experiments

In section A.3.1 and section A.3.2, we explore the dynamics effect in manipulators. Section A.3.3 shows the learning curves for 7-DOF robots with different link lengths and dynamics. In section A.3.4, we show more training details of HCP-E experiments on different combinations of robot types and how well HCP-E models perform on robots that belong to the same training robot types but with different link lengths and dynamics.

### A.3.1 Effect of Dynamics in Transferring Policies for Manipulation

Explicit encoding is made possible when knowing the dynamics of the system doesn't help learning. In such environments, as long as the policy network is exposed to a diversity of robots with different dynamics during training, it will be robust enough to adapt to new robots with different dynamics. To show that knowing ground-truth dynamics doesn't help training for reacher and peg insertion tasks, we experimented on 7-DOF robots (Type I) with different dynamics only with following algorithms:DDPG+HER, DDPG+HER+dynamics, DDPG+HER+random number. The first one uses DDPG+HER with only joint angles and joint velocities as the state. The second experiment uses DDPG+HER with the dynamics parameter vector added to the state. The dynamics are scaled to be within $[0, 1)$. The third experiment uses DDPG+HER with a random vector ranged from $[0, 1)$ added to states that is of same size as the dynamics vector. The dynamics parameters sampling ranges are shown in Table A.1. The number of training robots is 100.

Figure A.4 shows that DDPG+HER with only joint angles and joint velocities as states is able to achieve about 100% success rate in both reacher and peg insertion (fixed hole position) tasks. In fact, we see that even if state is augmented with a random vector, the policy network can still generalize over new testing robots, which means the policy network learns to ignore the augmented part. Figure A.4 also shows that with ground-truth dynamics parameters or random vectors input to the policy and value networks, the learning process becomes slower. In hindsight, this makes sense because the dynamics information is not needed for the policy network and if we forcefully feed in those information, it will take more time for the network to learn to ignore this part and train a robust policy across robots with different dynamics.

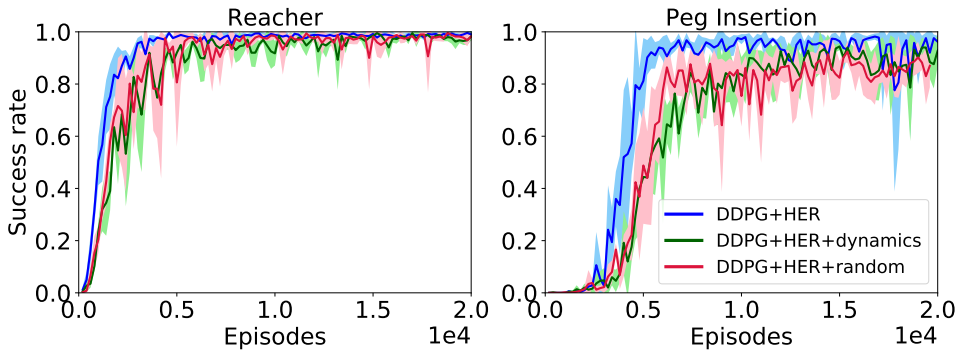



**Figure A.4:** Learning curves on 7-DOF robots with different dynamics only.

### A.3.2 How robust is the policy network to changes in dynamics?

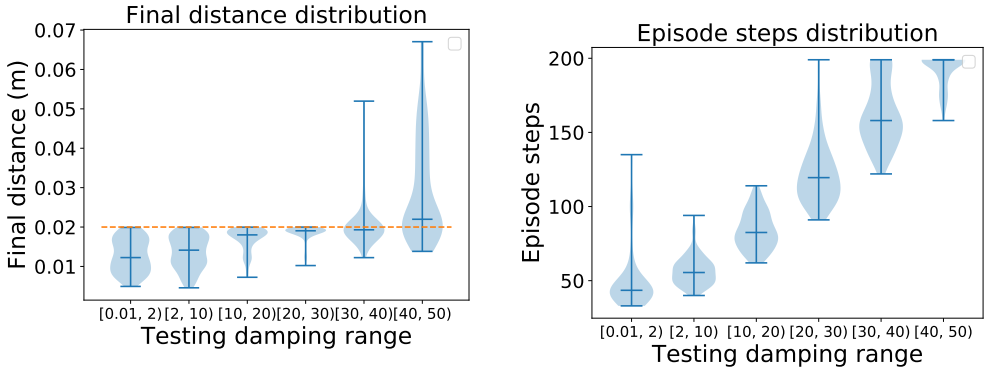
We performed a stress test on the generalization or robustness of the policy network to variation in dynamics. The experiments are similar to those in section A.3.1, but the training joint damping values are randomly sampled from $[0.01, 2)$ this time. Other dynamics parameters are still randomly sampled according to Table A.1. The task here is peg insertion. Figure A.5 and Table A.5 show the generalization capability of the DDPG + HER model with only joint angles and joint velocities as the state.

We can see from Figure A.5 and Table A.5 that even though the DDPG + HER model is trained with joint damping values sampled from $[0.01, 2)$, it can successfully control robots with damping values sampled from other ranges including

**Table A.5:** Success rate on 100 testing robots

| Testing damping range | Success rate |
|---|---|
| $[0.01, 2)$ | 100% |
| $[2, 10)$ | 100% |
| $[10, 20)$ | 100% |
| $[20, 30)$ | 100% |
| $[30, 40)$ | 85% |
| $[40, 50)$ | 47% |

$[2, 10), [10, 20), [20, 30)$ with 100% success rate. It is noteworthy that a damping value of 1 corresponds to critical damping (which is what most practical systems aim for), while ¡ 1 is under-damped and above is over-damped. For the damping range $[30, 40)$, the success rate is 85%. In damping range $[40, 50)$, the success rate is 47%. Note that each joint has a torque limit, so when damping becomes too large, the control is likely to be unable to move some joints and thus fail. Also, the larger the damping values are, the more time steps it takes to finish the peg insertion task, as shown in Figure A.5b.




**(a)** Distance to desired position at last step

**(b)** Episode length

**Figure A.5:** Performance (violin plots) on 100 testing robots with damping values sampled from different ranges using the DDPG+HER model trained with damping range $[0.01, 2)$). Other dynamics parameters are still randomly sampled according to Table A.1. The left plot shows the distribution of the distances between the robot's peg bottom and the target peg bottom position at the end of episode. The right plot shows the distribution of the episode length. An episode will be ended early if the peg is inserted into the hole successfully and the maximum number of episode time steps is 200. The three horizontal lines in each violin plot stand for the lower extrema, median value, and the higher extrema.

### A.3.3 Learning curves for 7-DOF robots with different link length and dynamics

In this section, we provide two supplementary experiments on training 7-DOF (type I) to perform reacher and peg insertion task.

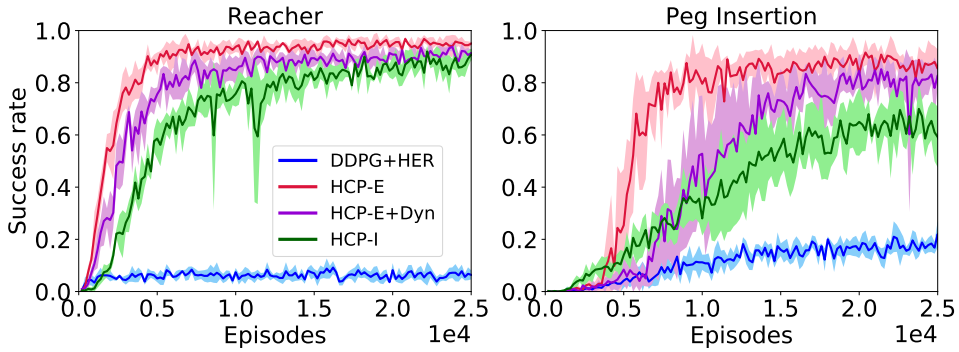
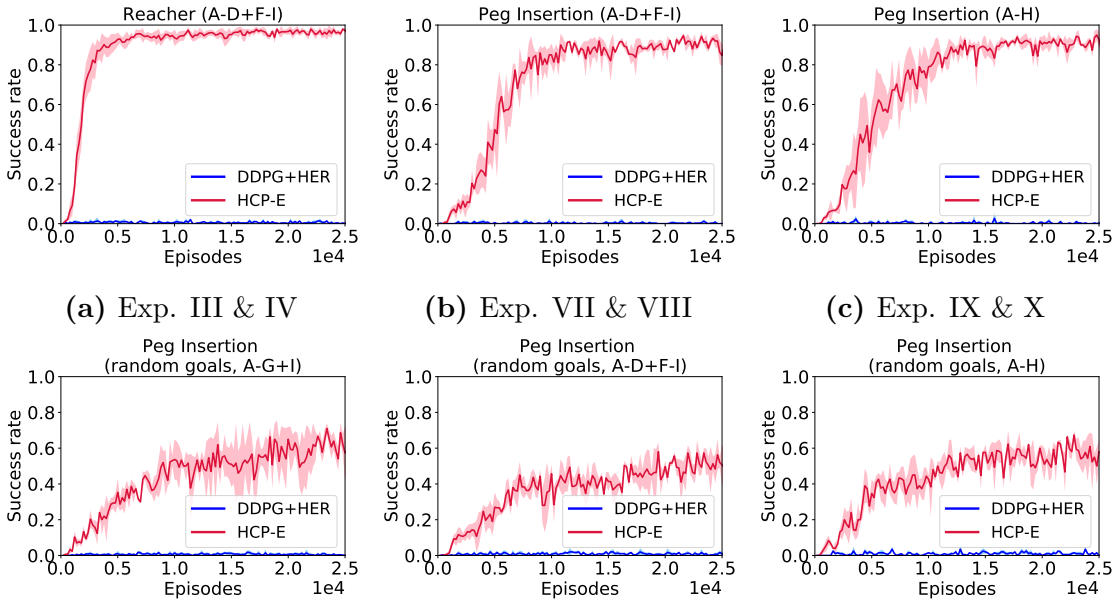



**Figure A.6:** Learning curves for 7-DOF robots with different link length and dynamics. We show the HCP-E+Dyn learning curves only for comparison. In real robots, dynamics parameters are usually not easily accessible. So it's not pratical to use dynamics information in robotics applications. We can see that both HCP-I and HCP-E got much higher success rates on both tasks than vanilla DDPG+HER.

### A.3.4 Multi-DOF robots learning curves

Figure A.7 provides more details of training progress in different experiments.




**(a)** Exp. III & IV **(b)** Exp. VII & VIII **(c)** Exp. IX & X

**(d)** Exp. XI & XII **(e)** Exp. XIII & XIV **(f)** Exp. XV & XVI

**Figure A.7:** Learning curves for multi-DOF setting. Symbol A,B,...,I in the figure represent the types of robot used in training. All these experiments are only trained on 8 types of robots (leave one out). The 100 testing robots used to generate the learning curves are from the same training robot types but with different link length and dynamics. The second row shows the results on peg insertion task with hole position randomly generated within a 0.2m box region. (a): reacher task with robot types A-D + F-I. (b): peg insertion task with a fixed hole position with robot types A-D + F-I. (c): peg insertion task with a fixed hole position with robot types A-H. (d): peg insertion task with a random hole position with robot types A-G + I. (e): peg insertion task with a random hole position with robot types A-D + F-I. (f): peg insertion task with a random hole position with robot types A-H.

Table 2.1 in the paper shows how well HCP-E models perform when they are applied to the new robot type that has never been trained before. Table A.6 to A.13 show how the universal policy behaves on the robot types that have been trained before. These robots are from the training robot types, but with different link lengths and dynamics.

The less DOF the robot has, the less dexterous the robot can be. Also, where to place the $n$ joints affects the workspace of the robot and determine how flexible the robot can be. Therefore, we can see some low success rate data even in trained robot types. For example,the trained HCP-E model only got 6.70 success rate when tested on robot type D which has actually been trained in peg insertion tasks with random

hole positions, as shown in Table A.11. This is because its joint displacements and number of DOFs limit the flexibility as shown in Figure 2.2d. Type D doesn't have joint $J_4$ and $J_5$ which are crucial for peg insertion tasks.

**Table A.6:** Zero-shot testing performance on training robot types (Exp. I & II)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | I |
| HCP-E | 93.10± 2.91 | 95.70± 1.55 | 98.20± 1.55 | 97.50± 1.02 | 95.30± 1.49 | 94.00± 3.26 | 98.40± 1.11 | 97.90± 1.67 |
| DDPG+HER | 1.00± 1.22 | 1.00± 1.00 | 2.50± 1.36 | 0.10± 0.30 | 0.70± 0.78 | 1.20± 1.40 | 1.30± 1.35 | 2.00± 1.26 |

**Table A.7:** Zero-shot testing performance on training robot types (Exp. III & IV)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | F | G | H | I |
| HCP-E | 92.00± 2.28 | 89.60± 3.01 | 98.60± 1.20 | 99.00± 0.63 | 96.70± 1.42 | 97.90± 1.64 | 99.30± 0.64 | 99.20± 0.60 |
| DDPG+HER | 1.30± 0.90 | 1.30± 0.89 | 1.60± 0.92 | 0.70± 0.46 | 0.20± 0.40 | 2.30± 1.18 | 0.90± 0.83 | 1.40± 0.92 |

**Table A.8:** Zero-shot testing performance on training robot types (Exp. V & VI)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | I |
| HCP-E | 91.10± 2.77 | 95.90± 1.92 | 98.50± 1.50 | 84.89± 3.29 | 94.70± 1.85 | 92.00± 2.97 | 97.20± 1.32 | 94.20± 2.79 |
| DDPG+HER | 0.30± 0.46 | 1.90± 1.30 | 3.00± 1.26 | 0.00± 0.00 | 0.00± 0.00 | 0.00± 0.00 | 0.60± 0.66 | 0.00± 0.00 |

**Table A.9:** Zero-shot testing performance on training robot types (Exp. VII & VIII)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | I |
| HCP-E | 88.60± 2.45 | 95.30± 2.00 | 98.90± 0.83 | 83.30± 3.49 | 81.30± 3.20 | 92.00± 3.13 | 89.40± 3.20 | 88.00± 4.54 |
| DDPG+HER | 3.30± 2.32 | 1.70± 1.00 | 0.00± 0.00 | 0.00± 0.00 | 0.00± 0.00 | 0.00± 0.00 | 0.00± 0.00 | 0.10± 0.30 |

**Table A.10:** Zero-shot testing performance on training robot types (Exp. IX & X)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H |
| HCP-E | 92.90± 3.59 | 95.90± 1.70 | 97.30± 1.10 | 90.90± 3.58 | 95.59± 1.43 | 94.60± 1.28 | 98.80± 0.60 | 97.10± 1.51 |
| DDPG+HER | 1.60± 1.20 | 2.30± 1.27 | 0.40± 0.66 | 0.00± 0.00 | 1.80± 1.54 | 0.00± 0.00 | 0.40± 0.49 | 0.00± 0.00 |

**Table A.11:** Zero-shot testing performance on training robot types (Exp. XI & XII)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | I |
| HCP-E | 71.00± 5.22 | 85.80± 3.19 | 89.00± 2.53 | 6.70± 2.53 | 79.30± 3.90 | 45.70± 4.86 | 88.50± 2.42 | 68.50± 5.18 |
| DDPG+HER | 1.70± 1.88 | 3.90± 2.20 | 0.90± 1.04 | 0.10± 0.30 | 2.50± 0.92 | 0.10± 0.30 | 1.00± 1.00 | 0.70± 0.78 |

**Table A.12:** Zero-shot testing performance on training robot types (Exp. XIII & XIV)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | F | G | H | I |
| HCP-E | 64.70± 6.30 | 86.10± 3.36 | 89.60± 2.95 | 54.10± 3.53 | 58.60± 3.83 | 83.20± 2.96 | 66.30± 3.57 | 62.50± 4.03 |
| DDPG+HER | 0.30± 0.46 | 0.10± 0.30 | 1.90± 0.94 | 0.00± 0.00 | 0.20± 0.40 | 1.90± 1.30 | 0.10± 0.30 | 2.80 ± 1.60 |

**Table A.13:** Zero-shot testing performance on training robot types (Exp. XV & XVI)

| Alg. | Testing Robot Types | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G | H |
| HCP-E | 73.70± 4.79 | 86.20± 4.04 | 80.90± 3.88 | 16.00± 3.26 | 69.70± 4.54 | 76.80± 3.25 | 85.80± 4.38 | 59.90± 5.22 |
| DDPG+HER | 1.90± 1.37 | 7.60± 2.65 | 3.10± 1.04 | 0.00± 0.00 | 3.70± 1.62 | 0.60± 0.66 | 1.30± 1.00 | 0.40 ± 0.80 |

# Appendix B

# Exploration Policies for Navigation

## B.1 SPL Metric

As defined by [81], Success weighted by (normalized inverse) Path Length or SPL is computed as follows. Here, $\ell_i$ is the shortest-path distance between the starting and goal position, $p_i$ is the length of the path actually executed by the agent, and $S_i$ is a binary variable that indicates if the agent succeeded. SPL is computed over $N$ trials as follows:

$$SPL = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{\ell_i}{\max(p_i, \ell_i)}. \tag{B.1}$$

## B.2 Examples of Policy Inputs and Maps

We show an example of the policy inputs (RGB image and maps in two different scales) at time $t$ and $t+10$ in Fig. B.1. Green area means the known(seen) traversable area, blue area means known non-traversable area, and the white area means unknown area. Fig. B.2 shows how the map evolves and the agent moves as it explores a new house.

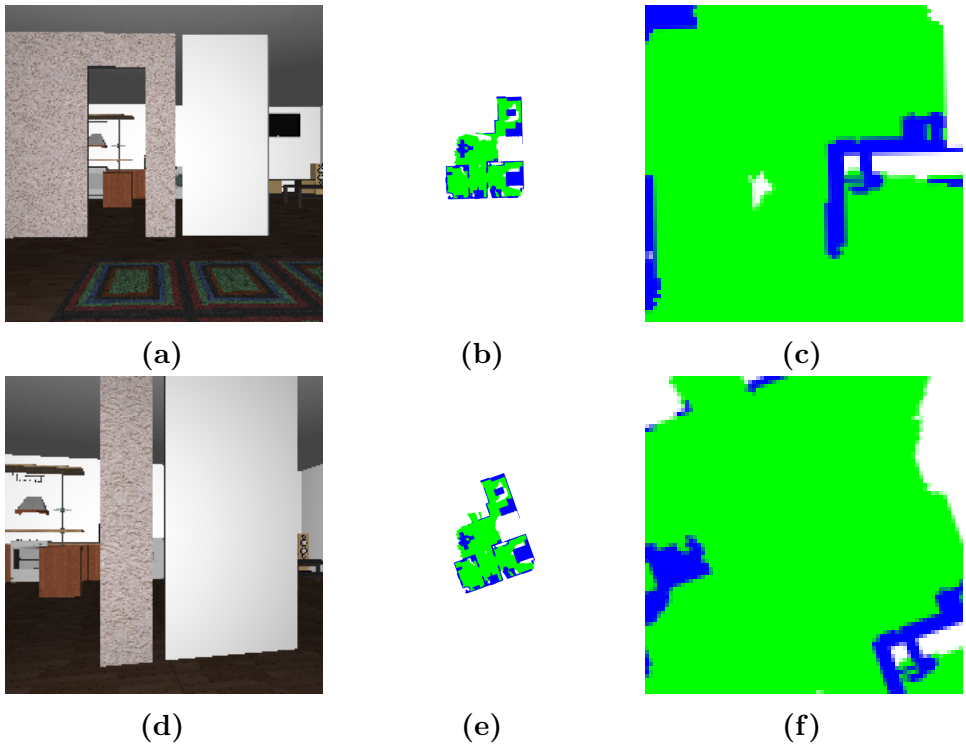



**Figure B.1:** Examples of policy inputs at time step $t$ and $t+10$. (a) and (d) are the RGB images, (b) and (e) are the coarse maps, and (c) and (f) are the fine maps.

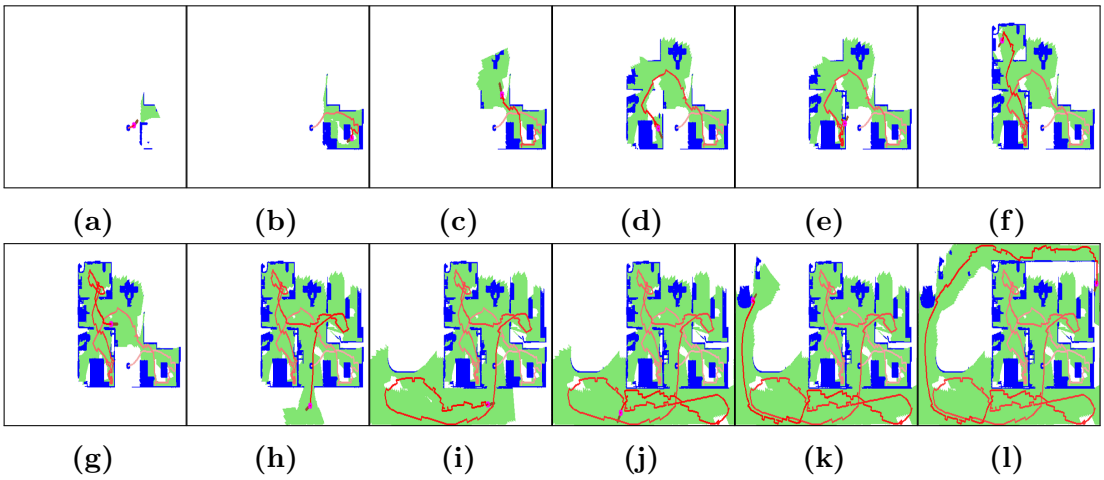



**Figure B.2:** Snapshots of the built map as the agent explores the house

## B.3 Experimental Details

### B.3.1 Environment Details

In this paper, we simulated the agent in House3D environment [77] and used 20 houses for training and 20 new houses for testing. Fig. C.1 shows the distribution of the total traversable area of these houses. Fig. C.2 shows some examples of the top-down views for training and testing houses (white is traversable, black is occupied).

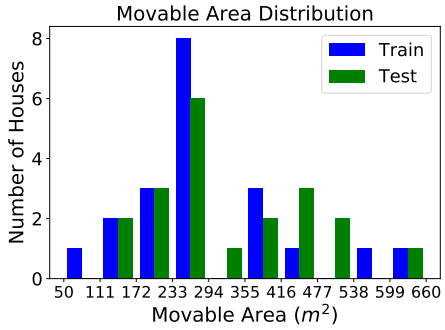



**Figure C.1:** Area distribution of training and testing houses (20 houses in each case). The average movable area of the training houses is 289.7 $m^2$. The average movable area of the testing houses is 327.9 $m^2$.

**Figure C.2:** Examples of house layouts (top-down views). **The left** three figures show 3 examples of the training houses. **The right** three figures show 3 examples of the testing houses. White regions indicate traversable area, black regions represent occupied area.

### B.3.2 Map Reconstruction

Given a sequence of camera poses (extrinsics) $([\boldsymbol{R}_1, \boldsymbol{t}_1], [\boldsymbol{R}_2, \boldsymbol{t}_2], ..., [\boldsymbol{R}_N, \boldsymbol{t}_N])$, and the corresponding depth images $(\boldsymbol{D}_1, \boldsymbol{D}_2, ..., \boldsymbol{D}_N)$ at each time step as well as the camera intrinsics $\boldsymbol{K} \in \mathbb{R}^{3\times3}$, we can reconstruct the point cloud of the scene based on the

principle of mulitple-view geometry [47]. We can convert the points from pixel space to the camera space and then to the world space. More specifically, the formulas to achieve this can be summarized as follows:

$$\lambda_{ij}\boldsymbol{x}_{ij} = \boldsymbol{K}\left[\boldsymbol{R}_i, \boldsymbol{t}_i\right]\boldsymbol{w}_{ij} \quad \forall j \in \{1, 2, ..., S\}, i \in \{1, ..., N\} \tag{B.2}$$

$$\boldsymbol{W} = \bigcup_{i=1}^{N}\bigcup_{j=1}^{S}\boldsymbol{w}_{ij} \tag{B.3}$$

where $S$ is the total number of pixels in each depth image, $\boldsymbol{x}_{ij} \in \mathbb{R}^3$ is the homogeneous coordinates for $j_{\text{th}}$ pixel on $i_{\text{th}}$ depth image $D_i$, $\lambda_{ij} \in \mathbb{R}$ is the depth value of the $j_{\text{th}}$ pixel on $i_{\text{th}}$ depth image $D_i$, and $\boldsymbol{w}_{ij} \in \mathbb{R}^4$ is the homogeneous coordinates for the corresponding point in the world coordinate system. We can get $\boldsymbol{w}_{ij}$ from $\boldsymbol{x}_{ij}$ based on Equation (B.2). And we can merge points via Equation (B.3).

### B.3.3 Training Details

The occupancy map generated by the agent itself uses a resolution of 0.05m. Our policy uses a coarse map, and a detailed map. The coarse map captures information about a 40m×40m area around the agent, at a resolution of of $0.5m$. The occupancy map is down-sampled from $800 \times 800$ to $80 \times 80$ to generate the coarse map that is fed into the policy network. The detailed map captures information about a 4m×4m area around the agent at a $0.05m$ resolution. RGB images are rendered at $224 \times 224$ resolution in House3D and re-sized into $80 \times 80$ before they are fed into the policy network.

The size of the last fully-connected layer in ResNet18 architecture is modified to 128. Outputs of the three ResNet18 networks are concatenated into a 384-dimensional vector. This is transformed into a 128-dimensional vector via a fully-connected layer. Next, it's fed into a single-layer RNN(GRU) layer with 128 hidden layer size. The output of RNN layer is followed by two heads. One head (the policy head) has two fully-connected layers $(128 - 32 - 6)$. The other head (the value head) has two fully-connected layers $(128 - 32 - 1)$. We use ELU as the nonlinear activation function for the fully-connected layers after ResNet18.

Each episode consists of 500 steps in training. RNN time sequence length is 20.

Coverage area $C(M_t)$ is measured by the number of covered grids in the occupancy map. Coefficient $\alpha$ for the coverage reward $R_{int}^{\text{cov}}(t)$ is 0.0005, coefficient $\beta$ for $R_{int}^{\text{coll}}(t)$ is 0.006. PPO entropy loss coefficient is 0.01. Network is optimized via Adam optimizer with a learning rate of 0.00001.

### B.3.4 Noise Model

Details of noise generation for experiments with estimation noise in Section 3.4.2:

1. Without loss of generality, we initialize the agent at the origin, that is $\hat{x}_0 = x_0 = \mathbf{0}$.
2. The agent takes an action $a_t$. We add truncated Gaussian noise to the action primitive(e.g., move forward 0.25m) to get the estimated pose $\hat{x}_{t+1}$, i.e., $\hat{x}_{t+1} = \hat{x}_t + \tilde{a}_t$ where $\hat{x}_t$ is the estimated pose in time step t and $\tilde{a}_t$ is the action primitive $a_t$ with added noise.
3. Iterate the second step until the maximum number of steps is reached.

Thus, in this noise model, the agent estimates its new pose based on the estimated pose from the last time step and the executed action. Thus, we dont use oracle odometry in the noise experiments. This noise model leads to compounding errors over time (as in the case of a real robot), though we acknowledge that this simple noise model may not perfectly match noise in the real world.

### B.3.5 Imitation Learning Details

We use behavioral cloning technique [82, 83, 84] to imitate the human behaviors in exploring the environments. We got the human demonstration trajectories from [74]. We ignored the question-answering part of the data and only used the exploration trajectories. We cleaned up the data by removing the trajectories that have less than 100 time steps. The trajectories are then converted into short sequences of trajectory segments $((s_i, a_i, s_{i+1}, a_{i+1}, ..., s_{i+T}, a_{i+T})$, where $T$ is based on the RNN sequence length). The policy is pretrained with behavioral cloning by imitating actions $(a_i, a_{i+1}, ..., a_{i+T})$ from the human demonstrations given the states $(s_i, s_{i+1}, ..., s_{i+T})$.

# Bibliography


[1] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. 1

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 1

[3] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872, 2019. 1

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. 1, 6

[5] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015. 1, 3, 6

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. 1

[7] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, 2017. 2, 19, 21, 22

[8] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. In *CVPR*, 2017. 2, 19, 21, 25

[9] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. In *ICLR*, 2017. 2, 19, 21, 22

[10] Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *ICLR*, 2018. 2, 19, 21, 22

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 3

[12] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 3

[13] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint arXiv:1710.06537*, 2017. 3, 4, 5

[14] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009. 4

[15] Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2169–2176. IEEE, 2017. 4, 5

[16] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4019–4026. IEEE, 2016. 4

[17] Lerrel Pinto and Abhinav Gupta. Learning to push by grasping: Using multiple tasks for effective learning. *ICRA*, 2017. 4

[18] Andrei Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. 2016. 4

[19] Adithyavairavan Murali, Lerrel Pinto, Dhiraj Gandhi, and Abhinav Gupta. CASSL: Curriculum accelerated self-supervised learning. *ICRA*, 2018. 4

[20] Rajeswaran Aravind, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. *ICLR*, 2017. 4, 5

[21] A Nilim and L El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. In *Operations Research*, pages 780–798, 2005. 4

[22] Ajay Mandlekar, Yuke Zhu, Li Fei-Fei, and Silvio Savarese. Adversarially robust

policy learning: Active construction of physically-plausible perturbations. *IROS*, 2017. 4, 5, 16

[23] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. 2017. 4, 5

[24] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017. 4

[25] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. 4

[26] Abhishek Gupta, Coline Devin, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Learning invariant feature spaces to transfer skills with reinforcement learning. *arXiv preprint arXiv:1703.02949*, 2017. 4

[27] Botond Bocsi, Lehel Csató, and Jan Peters. Alignment-based transfer learning for robot models. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–7. IEEE, 2013. 4

[28] Samuel Barrett, Matthew E Taylor, and Peter Stone. Transfer learning for reinforcement learning on a physical robot. In *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)*, 2010. 5

[29] Jeffrey Mahler, Jacky Liang, Sherdil Niyaz, Michael Laskey, Richard Doan, Xinyu Liu, Juan Aparicio Ojea, and Ken Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *RSS*, 2017. 5

[30] Mohamed K. Helwa and Angela P. Schoellig. Multi-robot transfer learning: A dynamical system perspective. 2017. 5

[31] Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. In *International Conference on Learning Representations*, 2018. 5

[32] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018. 5, 6

[33] Igor Mordatch, Kendall Lowrey, and Emanuel Todorov. Ensemble-cio: Full-body

dynamic motion planning that transfers to physical humanoids. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5307–5314. IEEE, 2015. 5

[34] J-JE Slotine and Li Weiping. Adaptive manipulator control: A case study. *IEEE transactions on automatic control*, 33(11):995–1003, 1988. 5

[35] Hae-Won Park, Koushil Sreenath, Jonathan Hurst, and J. W. Grizzle. System identification and modeling for mabel, a bipedal robot with a cable-differential-based compliant drivetrain. In *Dynamic Walking Conference (DW)*, MIT, July 2010. 6

[36] Umashankar Nagarajan, Anish Mampetta, George A Kantor, and Ralph L Hollis. State transition, balancing, station keeping, and yaw control for a dynamically stable single spherical wheel mobile robot. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 998–1003. IEEE, 2009. 6

[37] Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006. 6

[38] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 6

[39] Gregory Kahn, Tianhao Zhang, Sergey Levine, and Pieter Abbeel. Plato: Policy learning using adaptive trajectory optimization. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3342–3349. IEEE, 2017. 6

[40] Wenhao Yu, Jie Tan, C Karen Liu, and Greg Turk. Preparing for the unknown: Learning a universal policy with online system identification. In *RSS*, 2017. 6, 23

[41] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *CoRR*, abs/1707.01495, 2017. 7

[42] ROS URDF. http://wiki.ros.org/urdf. 7

[43] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012. 10

[44] Xijian Huo, Yiwei Liu, Li Jiang, and Hong Liu. Design and development of a 7-dof humanoid arm. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 277–282. IEEE, 2012. 10

[45] Zvi Artstein. Discrete and continuous bang-bang and facial spaces or: look for the extreme points. *SIAM Review*, 22(2):172–185, 1980. 12

[46] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. *Robotics: Science and systems VII*, page 73, 2012. 16

[47] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. 19, 21, 54

[48] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005. 19, 21

[49] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006. 19, 21

[50] Sebastian Thrun, Maren Bennewitz, Wolfram Burgard, Armin B Cremers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Charles Rosenberg, Nicholas Roy, Jamieson Schulte, et al. MINERVA: A second-generation museum tour-guide robot. In *ICRA*, 1999. 19

[51] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation (CIRA)*, 1997. 19, 20, 28

[52] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 2016. 21

[53] Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José Castellanos, and Arnaud Doucet. A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*, 2009. 21

[54] Henry Carrillo, Ian Reid, and José A Castellanos. On the comparison of uncertainty criteria for active slam. In *ICRA*, 2012. 21

[55] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *ICLR*, 2018. 21

[56] Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *IROS*, 2017. 22

[57] Fereshteh Sadeghi and Sergey Levine. CAD$^2$RL: Real single-image flight without a single real image. In *RSS*, 2017. 22

[58] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam: Learning to explore with external memory. *arXiv preprint*, 2017. 22

[59] Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *International Conference on Simulation of*

*Adaptive Behavior: From Animals to Animats*, 1991. 22

[60] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015. 22

[61] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 22, 28

[62] Justin Fu, John Co-Reyes, and Sergey Levine. $EX^2$: Exploration with exemplar models for deep reinforcement learning. In *NIPS*, 2017. 22

[63] Manuel Lopes, Tobias Lang, Marc Toussaint, and Pierre-Yves Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. In *NIPS*, 2012. 22

[64] Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *NIPS*, 2005. 22

[65] Dinesh Jayaraman and Kristen Grauman. Learning to look around: Intelligently exploring unseen environments for unknown tasks. In *CVPR*, 2018. 22

[66] Kai Xu, Lintao Zheng, Zihao Yan, Guohang Yan, Eugene Zhang, Matthias Niessner, Oliver Deussen, Daniel Cohen-Or, and Hui Huang. Autonomous reconstruction of unknown indoor scenes guided by time-varying tensor fields. In *SIGGRAPH Asia*, 2017. 22

[67] Shi Bai, Jinkun Wang, Fanfei Chen, and Brendan Englot. Information-theoretic exploration with bayesian optimization. In *IROS*, 2016. 22

[68] Thomas Kollar and Nicholas Roy. Trajectory optimization using reinforcement learning for map exploration. *IJRR*, 2008. 22

[69] Lennart Ljung. *System identification: theory for the user*. Prentice-hall, 1987. 23

[70] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. $RL^2$: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016. 23

[71] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018. 23

[72] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *ICLR*, 2018. 25

[73] Joao F Henriques and Andrea Vedaldi. MapNet: An allocentric spatial memory for mapping environments. In *CVPR*, 2018. 25

[74] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Embodied Question Answering. In *CVPR*, 2018. 26, 27, 28, 55

[75] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992. 26

[76] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 26, 28

[77] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. *arXiv preprint arXiv:1801.02209*, 2018. 27, 53

[78] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *CVPR*, 2017. 27

[79] Christian Dornhege and Alexander Kleiner. A frontier-void-based approach for autonomous exploration in 3D. *Advanced Robotics*, 2013. 28

[80] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. In *ICLR*, 2019. 28

[81] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir Zamir. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 33, 51

[82] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009. 55

[83] Ryszard S Michalski, Ivan Bratko, and Avan Bratko. *Machine learning and data mining; methods and applications*. John Wiley & Sons, Inc., 1998. 55

[84] Robert G Abbott. Behavioral cloning for simulator validation. In *Robot Soccer World Cup*, pages 329–336. Springer, 2007. 55