How to Train Your Robot - New Environments for Robotic Training and New Methods for Transferring Policies from the Simulator to the Real Robot

Florian Golemo

How to Train Your Robot

New Environments for Robotic Training and New Methods for Transferring Policies from the Simulator to the Real Robot

A DISSERTATION PRESENTED
BY
FLORIAN GOLEMO
TO
THE ECOLE DOCTORALE MATHÉMATIQUES ET INFORMATIQUE
UNIVERSITÉ DE BORDEAUX

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SUBJECT OF
INFORMATIQUE / COMPUTER SCIENCE

UNIVERSITE DE BORDEAUX
BORDEAUX, FRANCE

19TH DECEMBER 2018

Jury:

Pierre-Yves Oudeyer, INRIA Bordeaux - Directeur de thèse

Aaron Courville, MILA Montreal - Cosuperviseur

Stéphane Doncieux, ISIR Paris - Rapporteur

Alain Dutech, LORIA Nancy - Rapporteur

Vincent Lepetit, LABRI Bordeaux - Examinateur, Président du Jury

Vincent Padois, INRIA Bordeaux - Examinateur

Xavier Hinaut, INRIA Bordeaux - Invité

Thesis advisor: Pierre-Yves Oudeyer & Aaron Courville Florian Golemo

How to Train Your Robot

New Environments for Robotic Training and New Methods for Transferring Policies from the Simulator to the Real Robot

Abstract

Robots are the future. But how can we teach them useful new skills? This work covers a variety of topics, all with the common goal of making it easier to train robots.

The first main component of this thesis is our work on model-building sim2real transfer. When a policy has been learned entirely in simulation, the performance of this policy is usually drastically lower on the real robot. This can be due to random noise, to imprecisions, or to unmodelled effects like backlash. We introduce a new technique for learning the discrepancy between the simulator and the real robot and using this discrepancy to correct the simulator.

We found that for several of our ideas there weren't any suitable simulations available. Therefore, for the second main part of the thesis, we created a set of new robotic simulation and test environments. We provide (1) several new robot simulations for existing robots and variations on existing environments that allow for rapid adjustment of the robot dynamics.

We also co-created (2) the Duckietown AIDO challenge, which is a large scale live robotics competition for the conferences NIPS 2018 and ICRA 2019. For this challenge we created the simulation infrastructure, which allows participants to train their robots in simulation with or without ROS. It also lets them evaluate their submissions automatically on live robots in a "Robotarium".

In order to evaluate a robot's understanding and continuous acquisition of language, we developed the (3) Multimodal Human-Robot Interaction benchmark (MHRI). This test set contains several hours of annotated recordings of different humans showing and pointing at common household items, all from a robot's perspective. The novelty and difficulty in this task stems from the realistic noise that is included in the dataset: Most humans were non-native English speakers, some

objects were occluded and none of the humans were given any detailed instructions on how to communicate with the robot, resulting in very natural interactions.

After completing this benchmark, we realized the lack of simulation environments that are sufficiently complex to train a robot for this task. This would require an agent in a realistic house settings with semantic annotations. That is why we created (4) HoME, a platform for training household robots to understand language. The environment was created by wrapping the existing SUNCG 3D database of houses in a game engine to allow simulated agents to traverse the houses. It integrates a highly-detailed acoustic engine and a semantic engine that can generate object descriptions in relation to other objects, furniture, and rooms.

The third and final main contribution of this work considered that a robot might find itself in a novel environment which wasn't covered by the simulation. For such a case we provide a new approach that allows the agent to reconstruct a 3D scene from 2D images by learning object embeddings, since especially in low-cost robots a depth sensor is not always available, but 2D cameras a common. The main drawback of this work is that it currently doesn't reliably support reconstruction of color or texture. We tested the approach on a mental rotation task, which is common in IQ tests, and found that our model performs significantly better in recognizing and rotating objects than several baselines.

KEYWORDS: Deep Learning, Robotics, Sim2Real, Simulation

Directeur de thèse: Pierre-Yves Oudeyer & Aaron Courville Florian Golemo

Comment Dresser Votre Robot

Nouveaux Environnements de Formation Robotique et Nouvelles Méthodes de Transfert de Stratégies du Simulateur au Vrai Robot

Abstract

Les robots sont l'avenir. Mais comment pouvons-nous leur apprendre de nouvelles compétences utiles? Ce travail couvre une variété de sujets, ayant tous pour but commun de faciliter l'entraînement des robots.

La première composante principale de cette thèse est notre travail sur le transfert de modélisation sim2real. Lorsqu'une stratégie a été entièrement apprise en simulation, ses performances sont généralement considérablement inférieures à celles du vrai robot. Cela peut être dû à du bruit aléatoire, à des imprécisions ou à des effets non modélisés, tels que des réactions en retour. Nous introduisons une nouvelle technique pour apprendre la différence entre le simulateur et le vrai robot et pour l'utiliser afin de corriger le simulateur.

Nous avons constaté que pour plusieurs de nos idées, aucune simulation appropriée n'était disponible. Par conséquent, pour la deuxième partie principale de la thèse, nous avons créé un ensemble de nouvelles simulations robotiques et de nouveaux environnements de test. Nous fournissons (1) plusieurs nouvelles simulations pour des robots existants, ainsi que des variantes d'environnements existants, qui permettent un ajustement rapide de la dynamique du robot.

Nous avons également co-créé (2) le défi AIDO de Duckietown, qui est un concours de robotique en direct à grande échelle pour les conférences NIPS 2018 et ICRA 2019. Pour ce défi, nous avons créé l'infrastructure de simulation, qui permet aux participants d'entraîner leurs robots en simulation avec ou sans ROS. Il leur permet également d'évaluer automatiquement leurs soumissions sur des robots en direct dans un "Robotarium".

Afin d'évaluer la compréhension et l'acquisition continue de langage par un robot, nous avons développé le (3) Test d'Interaction Multimodal Homme-Robot (MHRI). Cet ensemble de tests contient plusieurs heures d'enregistrements annotés de différentes personnes montrant et pointant des

objets ménagers courants, le tout du point de vue d'un robot. La nouveauté et la difficulté de cette tâche découlent du bruit réaliste inclus dans le jeu de données: la plupart des personnes n'était pas de langue maternelle anglaise, certains objets étaient obstrués et personne n'avait reçu d'instructions détaillées sur la manière de communiquer avec le robot, entraînant des interactions très naturelles.

Nous avons constaté un manque flagrant de simulations d'environnements domestiques réalistes, avec annotations sémantiques, qui permettraient à un agent d'acquérir les compétences nécessaires pour maîtriser une telle tâche. C'est pourquoi nous avons créé (4) HoME, une plate-forme de formation de robots domestiques à la compréhension du langage. L'environnement a été créé en encapsulant la base de données existante SUNCG 3D, composée de maisons, dans un moteur de jeu pour permettre aux agents simulés de parcourir ces dernières. Il intègre un moteur acoustique très détaillé et un moteur sémantique pouvant générer des descriptions d'objets en relation avec d'autres objets, meubles et pièces.

La troisième et dernière contribution principale de ce travail prend en considération le fait qu'un robot peut se trouver dans un nouvel environnement non couvert par la simulation. Dans un tel cas, nous fournissons une nouvelle approche qui permet à l'agent de reconstruire une scène 3D à partir d'une seule image 2D en apprenant l'intégration d'objets. Le principal inconvénient de ce travail est qu'il ne prend actuellement pas en charge de manière fiable la reconstruction de couleur et de texture. Nous avons testé cette approche sur une tâche de rotation mentale, courante dans les tests de QI, et avons constaté que notre modèle arrivait nettement mieux à reconnaître et à faire pivoter des objets que plusieurs modèles de référence.

MOTS-CLES: Deep Learning, Robotics, Sim2Real, Simulation

Contents

Dedicated to my fiancee, our future kids and our future corgi.

Acknowledgments

The following acknowledgements are in no particular order. I would like to thank the IGLU / CHIST-ERA project for funding my studies, Jean Rouat for setting up the project, and all the other IGLU members for the great collaborations and support, especially Pablo Azagra for the collaboration on the MHRI project and Florian Strub for the discussions and input. I'd also like to thank Stephane Doncieux for helping me find this position in the first place. I think I alreaady mentioned her in the dedication, but I'd still like to thank my fiancee for keeping me alive and well-nourished, especially over the last months of my thesis, for keeping me on track and for great ideas that contributed to this thesis. Of course, I'd like to thank my parents, especially for helping me move to France and for every once in a while helping me out when the university couldn't figure out how to pay me for teaching.

Big thanks to both my supervisors, Pierre-Yves and Aaron, for the outstanding support. Special thanks to Pierre-Yves for always challenging our ideas, for making us strive for more robust algorithms, and for providing the robots and tools without this work would not have been possible. He was also there for me when I was down about experiments not working and was open to new research directions. Also thanks to Aaron for offering his co-supervision and a great internship that made much of the work that is presented in this thesis possible. While I'm talking about the labs, I'd like to give a shout out to the FLOWERS team, especially William Schueller for all the help setting up shop in Bordeaux and dealing with the administration, Baptiste Busch for all the coffee breaks and the great discussions, and Theo Segonds for the help with the robots. Also the MILA members, especially Adrien Ali Taiga for going through with the sim2real project even though we had our ups and downs, Cesar Laurent for translating the abstract, Michael Noukhovitch for donating his mate tea and gurd - your caffeine has greatly contributed to these lines, Ankesh Anand and Ethan Perez for their collaboration on the HoME project, Nithin Vasisth for making the both of us go through a 4 part lecture series on sim2real that made me re-read all the papers and gave me a bit better overview over the field, and Sai Rajeswar and Fahim Mannan for their collaboration on the Pix2Scene project. And thanks to Derek Nowrouzezahrai for all the great discussions over the Pix2Scene project. Also huge thanks to Liam Paull for letting me join the Duckietown community, for being able to complain about the administration together, and for his input on this thesis. Also thanks to all other members of Duckietown, especially Bhairav Mehta for taking a lot of work off my shoulders during

these months, and Jacopo Tani for help with the physical Duckiebots and for hosting me twice.

This is getting really long and probably nobody will ever read past here, but I have two more. Very warm thanks to David Vazquez from Element AI for letting me make a sandwich there every once in a while, for helping me move to Canada, and for the general carreer advice, not to mention his help and advice on the Pix2Scene project. And lastly I'd like to mention my former mentor, Evert Haasdijk, whose dissertation I shamelessly took some inspiration from for formatting and organizing this work.

0 Introduction

0.1 Background

AI has become an integral part of many systems that we use on a daily basis. Smartphones have voice assistants. Facebook recognizes faces in images and suggests who they belong to. On your commute you can use smart routing apps that predict traffic and save time. Suspicious emails are filtered into the spam folder. The rise of AI has been gradually building up, with its first attempts after world war 2, and its exponential rise in the mid-21st century. With the development of smarter algorithms, there has also been an increase in the development of robots. Pick & place robots are present in many factories, especially in the automotive industry; Lego Mindstorms is being used to playfully teach children the basics of robotics; and Roombas have made their way into many homes. As of writing, self-driving cars are on the horizon and medical, especially surgical, robots are gaining momentum. There is no reason to believe the spread of robots has peaked yet. We believe that in order to make the most out of this technology and its increasing adoption, it is important to educate more people in robotics; to teach robots new skills, and adapt them to new problems.

This thesis revolves around exactly this - how do you train your robot? In the industry, the robot's body and software are usually designed together for a very specific purpose, with high requirements in precision. These industrial systems are costly, very specialized, and consequently unsuitable for experimentation. To teach robotics and to prototype new behaviors, it is usually easier to resort to **simulations and low-cost robots**.

To teach a robot a new "trick", the following are required:

1. **A robot.** This can be any kind of robot, but in this work we focus on low-cost robots since they are easier to acquire, easier to experiment on, but harder to train. They often come with small mechanical imprecisions that are hard to model. However, if a robot training method works on low-cost robots, it's certain to also work on industrial robots. For the purpose of this thesis, we assume this is a given.

2. **A method for learning a skill** - either through imitation or through rewards/punishments. In this work we assume this too as given.

3. **A method for the robot to perceive the world.** Real-life robots might find themselves in new places, especially indoors, for which no map exists. This is somewhat easy to handle with expensive 3D sensors but since this work focuses on a low-cost approach, we're asking how this can be solved with conventional cameras.

4. **A simulation.** Most contemporary skill-acquisition methods contain exploration phases during which the agent explores new actions and their impact on the environment. If the learning is carried out on the real robot, this exploration can lead to the robot damaging itself or its surroundings. Also, many contemporary methods rely on deep neural networks. These require less structured input, but more samples (i.e. they run for a long time). To mitigate both the speed and damage aspect of skill acquisition, it's common to train in simulation. Simulations are easier to parallelize than real robot experiments and they allow for resetting the environment and the robot without any issues. The only remaining issue is finding a suitable simulation for a given task, one that contains the aspects that are relevant to the experiment. One part of this work consists of creating new simulations or modifying existing ones to accommodate a wider variety of training scenarios.

5. **A method of transferring the learned skill from simulation to the real robot (*sim2real*).** Policies that were learned in simulation often don't work flawlessly on the real robot due to noise, incorrect simulation parameters, or unmodelled robot properties. We also address this issue by providing a new way of easing the transfer from simulation to the real robot.

0.2 Structure & Contributions

Out of the requirements described in the previous section, this work is addressing #3-5 in reverse order: We start by assuming we have everything we need, and only a sim2real method is missing, go one step lower and investigate what kind of new simulations we require. Then we conclude with a contribution to the 3D perception problem:

Chapter 1 focuses on the sim2real transfer problem. Section 1.1 we provide a **new method for sim2real transfer** (Golemo, Ali Taiga, Oudeyer, & Courville, 2018). Through recording random movements on a real robot, mimicking these in simulation and recording the discrepancies, we can learn a "discrepancy model". This model can be used to correct the simulator. We demonstrate, that the policies learned in this augmented simulator transfer significantly better to the real robot. In many cases they perform close to the policy that was learned directly on the real robot. Chapter 1.2 extends this idea by **adding intelligent exploration methods**. When the initial real robot recording covers a wider range of states, the discrepancy model is more accurate for a wide range of tasks. Consequently, the policies learned in the augmented simulator perform better on the real robot than those that had the model learn through random exploration.

Chapter 2 focuses on simulators and datasets. Section 2.1 starts by detailing the **simulators** that we developed for the project outlined in the previous chapter. In section 2.2, we take a detour through an educational robot platform and discuss a **robot training challenge** that we co-developed over the course of these doctoral studies (Censi et al., 2018). After that, section 2.3 introduces a **large test dataset** that we created for testing robot assistants and assess their learning capabilities (Azagra et al., 2017). In the aftermath of this work, we learned that at the time there didn't exist any simulators that were sufficient to train such a robot. So we set out to build such a **home simulator platform**, which is presented in section 2.4 (Brodeur et al., 2018).

Chapter 3 contains our last contribution; a new method for **acquiring 3D models of the environment from single 2D images** by learning shape embeddings (Rajeswar et al., 2019). We trained this model without supervision, which makes this method more versatile and suitable to larger and more realistic environments, in which a ground truth 3D model isn't available.

Chapter 4 concludes this work with a discussion of some of the shortcomings and gaps between contributions that still need to be filled, and finally, the conclusion section lists our currently ongoing research and what future directions could be taken based on our works.

This thesis is based on the following papers:

Azagra, P., Golemo, F., Mollard, Y., Lopes, M., Civera, J., & Murillo, A. C. (2017). A multimodal dataset for object model learning from natural human-robot interaction. *Intelligent Robots and Systems (IROS) 2017 IEEE/RSJ International Conference*.

Brodeur, S., Perez, E., Anand, A., Golemo, F., Celotti, L., Strub, F., ... Courville, A. (2018). Home: A household multimodal environment. *6th International Conference on Learning Representations (ICLR) 2018, Workshop Track*.

Censi, A., Paull, L., Tani, J., Zilly, J., Ackermann, T., Beijbom, O., ... Frazzoli, E. (2018). The ai driving olympics at nips 2018. *Robotics: Science and Systems (RSS) 2018, Workshop Track*.

Golemo, F., Ali Taiga, A. A., Oudeyer, P.-Y., & Courville, A. (2018). Sim-to-real transfer with neural-augmented robot simulation. *Conference on Robot Learning (CoRL)*.

Rajeswar, S., Mannan, F., Vazquez, D., Golemo, F., Nowrouzezahrai, D., & Courville, A. (2019). Pix2scene: Learning implicit 3d representations from images. *7th International Conference on Learning Representations (ICLR) 2019*.

Well, dreams, they feel real while we're in them right?
Its only when we wake up then we realize that something was actually strange.

Christopher Nolan (from "Inception")

1 Back to the Reality

Let's transfer, baby. In this chapter we discuss why a sophisticated sim2real transfer method is necessary in the first place, the advantages and shortcomings of existing methods, and we introduce our own method; the task-independent neural-augmented simulator (NAS). We demonstrate the basic functionality on several (simulated and real) robots, before demonstrating the reliance on exploration in the first phase of the technique. We conclude the chapter by showing how the overall performance of this method can be improved via autonomous exploration.

1.1 Sim-to-Real Transfer with Neural-Augmented Robot Simulation

Despite the recent successes of deep reinforcement learning, teaching complex motor skills to a physical robot remains a hard problem. While learning directly on a real system is usually impractical, doing so in simulation has proven to be fast and safe. Nevertheless, because of the "reality gap," policies trained in simulation often perform poorly when deployed on a real system. In this work, we introduce a method for training a recurrent neural network on the differences between simulated and real robot trajectories and then using this model to augment the simulator. This Neural-Augmented Simulation (NAS) can be used to learn control policies that transfer significantly better to real environments than policies learned on existing simulators. We demonstrate the potential of our approach through a set of experiments on the Mujoco simulator with added backlash and the Poppy Ergo Jr robot. NAS allows us to learn policies that are competitive with ones that would have been learned directly on the real robot.

This section 1.1 is an extended version of our work that was published as

Florian Golemo, Adrien Ali Taiga, Pierre-Yves Oudeyer, Aaron Courville. **Sim-to-Real Transfer with Neural-Augmented Robot Simulation**. In Conference on Robot Learning (CoRL) 2018.

To this work we contributed: Modifying existing simulator environments to have flexible dynamics (see chapter 2.1 for more details); measuring the Poppy Ergo Jr. robot; and creating an entirely new simulation for this robot in PyBullet, after initially using the V-Rep simulator and finding that it couldn't handle resetting the internal state; creating the real robot experiments including the design of the task and the utensils; creating reusable code for high-frequency low-latency live control of the Poppy Ergo Jr. (also in chapter 2.1); coming up with the electro-conductive paint solution for contact detection; supervising all of the robot experiments; setting up one of the three sim2sim experiments and assisting in running and analyzing the other experiments; designing the neural networks for training the discrepancy model; contributing some of the code for the initial data collection, most of the code for the training of the discrepancy model, and the application to the simulator state; a modified policy training procedure based on the augmented simulator using an existing PPO implementation; most of the diagrams and tables in this article; and roughly half of the writing.

1.1.1 Introduction

Reinforcement learning (RL) with function approximation has demonstrated remarkable performance in recent years. Prominent examples include playing Atari games from raw pixels (Mnih et al., 2015), learning complex policies for continuous control (Schulman, Levine, Abbeel, Jordan, & Moritz, 2015; Lillicrap et al., 2015), and surpassing human performance on the game of Go (Silver et al., 2016). However, most of these successes were achieved in non-physical environments: simulations, video games, etc. Learning complex policies on physical systems remains an open challenge. Typical reinforcement learning methods require a large amount of interaction with the environment and, in addition, it is also often impractical, if not dangerous, to roll out a partially trained policy. Both these issues make it unsuitable for many tasks to be trained directly on a physical robot.

The fidelity of current simulators and rendering engines provides an incentive to use them as training grounds for control policies, hoping the newly acquired skills would transfer back to reality. However, this is not as easy as one might hope since no simulator perfectly captures reality. This problem is widely known as the *reality gap*. See Figure 1.1 for an illustration.

The reality gap can be seen as an instance of a *domain adaptation* problem, where the input distribution of a model changes between training (in simulation) and testing (in the real world). In the case of image classification and off-policy image-based deep reinforcement learning, this issue has sometimes been tackled by refining images from the *source domain* – where the training happens – to appear closer to images from the *target domain* – where evaluation is carried out (Shrivastava et al., 2017; Bousmalis et al., 2018).

In this work, we take a similar approach and apply it to continuous control. We use data collected from a real robot to train a recurrent neural network to predict the discrepancies between simulation and the real world. This allows us to improve the quality of our simulation by grounding it in realistic trajectories. We refer to our approach as *Neural-Augmented Simulation* (NAS). We can use it with any reinforcement learning algorithm, combining the benefits of simulation and fast offline training, while still learning policies that transfer well to robots in real environments. Since we collect data using a non-task-specific policy, NAS can be used to learn policies related to different tasks. Thus, we believe that NAS provides an efficient and effective strategy for multi-task sim-to-real transfer.

With our NAS strategy, we aim to achieve the best of both modeling modalities. While the recurrent neural network compensates for the unrealistically modeled aspects of the simulator; the simulator allows for better extrapolation to dynamic regimes that were not well explored under the

data collection policy. Our choice to use a recurrent model* is motivated by the desire to capture deviations that could violate the standard Markov assumption on the state space.

We evaluate our approach on two OpenAI Gym (Brockman et al., 2016) based simulated environments with an artificially created reality gap in the form of added backlash. We also evaluate on the Poppy Ergo Jr robot arm, a relatively inexpensive arm with dynamics that are not well modeled by the existing simulator. We find that NAS provides an efficient way to learn policies that approach the performance of policies trained directly on the physical system.

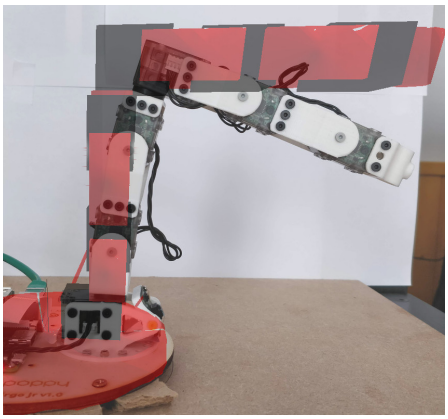


Figure 1.1: Impact of backlash: when setting both the simulated robot (red) and the real robot (white) to the resting position, the small backlash of each joint adds up to a noticeable difference in the end effector position.

1.1.2 Related work

Despite the recent success of deep reinforcement learning, learning on a real robot remains impractical due to poor sample efficiency. Classical methods for sim-to-real transfer include co-evolving the policy and the simulation (Zagal, Ruiz-del-Solar, & Vallejos, 2004; Bongard & Lipson, 2004), selecting for policies that transfer well (Koos, Mouret, & Doncieux, 2010; Zagal & Ruiz-Del-Solar, 2007), learning a transfer function of policies (Higuera, Meger, & Dudek, 2017), and modeling different levels of noise (Jakobi, Husbands, & Harvey, 1995). Nonetheless, these methods are usually limited to domains with a low-dimensional state-action space.

An alternative is to try to learn a forward model of the dynamics, a mapping between the current state and an action to the next state (Punjani & Abbeel, 2015; Fu, Levine, & Abbeel, 2016; Mordatch,

*We chose to use a Long Short-Term Model (LSTM) (Hochreiter & Schmidhuber, 1997) as our recurrent neural network.

Mishra, Eppner, & Abbeel, 2016). However, despite some successes, learning a forward model of a real robot remains an open challenge. The compounding error of these models quickly deteriorates over time and corrections are needed to compensate for the uncertainty of the model (Nagabandi, Kahn, Fearing, & Levine, 2017). These methods are also usually sensitive to the data used to train the model, which is often different from the state distribution encountered during policy learning (Deisenroth & Rasmussen, 2011; Finn, Levine, & Abbeel, 2016)

Domain adaptation has received a lot of focus from the computer vision community. There have been many approaches such as fine-tuning a model trained on the source domain using data from the target domain (Yosinski, Clune, Bengio, & Lipson, 2014), enforcing invariance in features between data from source and target domain (Ganin et al., 2016) or learning a mapping between source and target domain (Taigman, Polyak, & Wolf, 2016).

Similar ideas have been applied in RLx, such as previous work that focused on learning internal representations robust to change in the input distribution (Tzeng, Hoffman, Zhang, Saenko, & Darrell, 2014). Using data generated from the simulation during training was also used in bootstrapping the target policy (James & Johns, 2016), Progressive Neural Networks (Rusu et al., 2016) were also used to extend a simulation-learned policy to different target environments, but the most common method remains learning the control policy on the simulator before fine tuning on the real robot. Another quite successful recent method consists of randomizing all task-related physical properties of the simulation (Josh Tobin et al., 2017; Peng, Andrychowicz, Zaremba, & Abbeel, 2017). This approach is very time-consuming because in order for a policy to generalize to all kinds of inputs, it has to experience a very wide range of noise during learning.

Recently an approach very similar to our has been developed in the work of (Hanna & Stone, 2017). However, rather than learning a state space transition, the authors learn a transformation of actions. This learning is done on-policy so once the model is learned, the simulator augmentation doesn't transfer to other tasks.

In summary, existing approaches struggle with either computational complexity or with difficulties in transferability to other tasks. We would like to introduce a technique that is capable of addressing both these issues.

1.1.3 Method

1.1.3.1 Notation

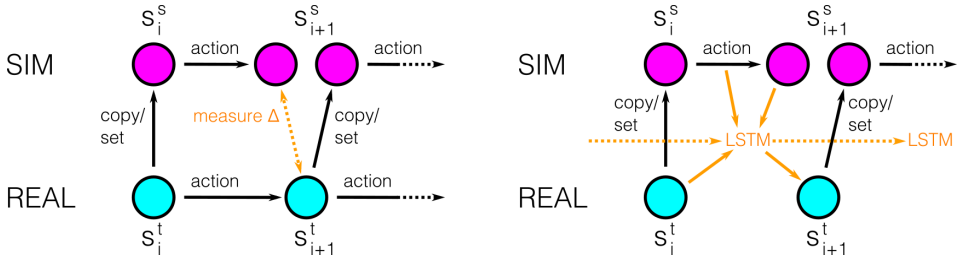
We consider a domain adaptation problem between a source domain D_s (usually a simulator) and a target domain D_t (typically a robot in the real world). Each domain is represented by a Markov

Decision Process (MDP) $\langle S, \mathcal{A}, p, r, \gamma \rangle$, where S is the state space, $\mathcal{A}$ the action space, for $s \in S$ and $a \in S, p(.|s, a)$ the transition probability of the environment, r the reward and γ the discount factor. We assume that the two domains share the same action space, however, because of the perceptual-reality gap state space, dynamics and rewards can be different even if they are structurally similar, we write $D_s = \langle S_s, \mathcal{A}, p_s, r_s \rangle$ and $D_t = \langle S_t, \mathcal{A}, p_t, r_t \rangle$ for the two MDPs. We also assume that rewards are similar (i.e $r_s = r_t$) and, most importantly, we assume that we are given access to the source domain and can reset it to a specific state. For example, for transfer from simulation to the robot, we assume we are free to reset the joint positions and angular velocities of the simulated robots.

1.1.3.2 Target domain data collection

We model the difference between the source and target domain, learning a mapping from trajectories in the source domain to trajectories in the target domain. Given a behavior policy μ (which could be random or provided by an expert) we collect trajectories from the target environment (i.e. real robot) follow actions given by μ.

1.1.3.3 Training the model



(a) Training phase **(b)** Policy Learning Phase

Figure 1.2: Left: Overview of the method for training the forward dynamics model. By gathering state differences when running the same action in simulation and on the real robot. Right: When the forward model is learned, it can be applied to simulator states to get the corresponding real state. This correction model (Δ) is time-dependent (implemented as LSTM). The policy learning algorithm only has access to the "real" states.

To train the LSTM in NAS, we sample an initial state $s_0 \sim p_t(s_0)$ from the target domain distribution and set the source domain to start from the same initial state (i.e $s_0^s = s_0^t = s_0$). At each time step an action is sampled following the behavior policy $a_i \sim \mu(s_i^t)$, then executed on the two domains to get the transition $(s_i, a_i, s_{i+1}^s, s_{i+1}^t)$. The source domain is then reset to the target domain state and the procedure is repeated until the end of the episode. The resulting trajectory

$\tau = (s_0, a_0, s_1^s, s_1^t, ..., s_{T-2}^s, a_T, s_{T-1}^s, s_{T-1}^t)$ of length T is then stored, the procedure is described in Algorithm 1.

After collecting the data the model φ, an LSTM (Hochreiter & Schmidhuber, 1997), is trained to predict s_{i+1}^t. The difference between two states is usually small, so the network outputs a correction term $\varphi(s_i^t, a_i, h, s_{i+1}^s) = s_{i+1}^t - s_{i+1}^s$ where h is the hidden state of the network. We also compare our approach with a forward model trained without using information from the source domain, $\psi(s_i, a_i, h) = s_{i+1}^t - s_i$. We normalize the inputs and outputs, and the model is trained with maximum likelihood using Adam (Diederik P Kingma & Ba, 2015).

For the Pusher, the neural network architecture is a fully connected layers with 128 hidden units with ReLU activations followed by a 3 layers LSTM with 128 hidden units and an other fully connected layers for the outputs. The Striker shares the same architecture with 256 hidden units instead. The ErgoReacher environment only needed an LSTM of 3 layers with 100 hidden units each. Networks are trained using the Adam optimizer for 250 epochs with a batch size of 1 and a learning rate of 0.01.

We used the PPO implementation and the recommended parameters for Mujoco from https://github.com/ikostrikov/pytorch-a2c-ppo-acktr.

Algorithm 1 Offline Data Collection

```
Initialize model φ
for episode= 1, M do
  Initialize the target domain from a state s_1^t
  Initialize the source domain to the same state s_1^s = s_1^t
  for i = 1, T do
    Select action a_i ~ μ(s_i^t) according to the behavior policy
    Execute action a_i on the source domain and observe new state s_{i+1}^s
    Execute action a_i on the target domain and observe new state s_{i+1}^t
    Update φ with the tuple s_{i+1}^s = s_{i+1}^t
    Set s_{i+1}^s = s_{i+1}^t
```

1.1.3.4 Transferring to the target domain

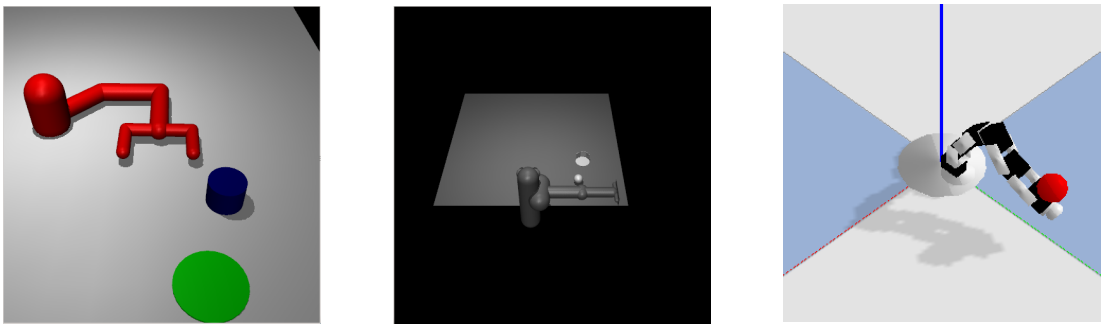
Once the model φ is trained, it can be combined with the source environment to learn a policy that will later be transferred to the target environment. We use PPO* (Schulman, Wolski, Dhariwal, Radford, & Klimov, 2017) in all our experiments but any other reinforcement learning algorithm could be used. During learning at each time step the current state transition in the source domain is passed

*A list of all our hyperparameters is included in the supplementary material.

to φ to compute an estimate of the current state in the target environment, an action a_i is chosen according to this estimate $a_i \sim \pi(\varphi(s_i^s, s_{i+1}^s))$, then the source domain state is set to the current estimate of the target domain state $\sim \varphi(s_i^s, s_{i+1}^s)$. This will allow our model to correct trajectories from the source domain, making them closer to the corresponding trajectory in the target domain.

Algorithm 2 Policy Learning in Source Domain

```
Given a RL algorithm 𝔸
A model φ
Initialize 𝔸
for episode= 1, M do
   Initialize the source domain from a state s_1^s
   Initialize the φ latent variables h
   for i = 1, T do
      Sample action a_i ~ π(s_i^s) using the behavioral policy from 𝔸
      Execute action a_i in the source domain, observe a reward r_i and a new state s_{i+1}^s
      Set ŝ_{i+1}^t = s_{i+1}^s + φ(s_i, a_i, h, s_{i+1}^s) the estimate of s_{i+1}^t given by φ
      Do a one-step policy update with 𝔸 using the transition (s_i^s, a_i, ŝ_{i+1}^t, r_i)
      Set the source domain to s_{i+1}^s = ŝ_{i+1}^t
```



(a) Pusher **(b)** Striker **(c)** ErgoReacher

Figure 1.3: Benchmark simulation environments used

1.1.4 Experiments

We evaluated our method on a set of simulated robotic control tasks using the MuJoCo physics engine (Todorov, Erez, & Tassa, 2012) as well as on the open-source 3D-printed physical robot "Poppy Ergo Jr" (Lapeyre et al., 2014) (see https://www.poppy-project.org/en/robots/poppy-ergo-jr)*.

*The code for our experiments can be found at https://github.com/aalitaiga/sim-to-real/

1.1.4.1 Simulated Environments Overview

We created an artificial reality gap using two simulated environments. The source and target environments are identical, except that the target environment has backlash. We also experimented with small changes in joints and link parameters but noted that it did not impact our results. Overall, the difference in backlash between the environments was significant enough to prevent policies trained on the source environment from performing well on the target environment (though an agent could solve the target environment if trained on it).

We used the following simulator environments for our experiments (more details on those in chapter 2.1):

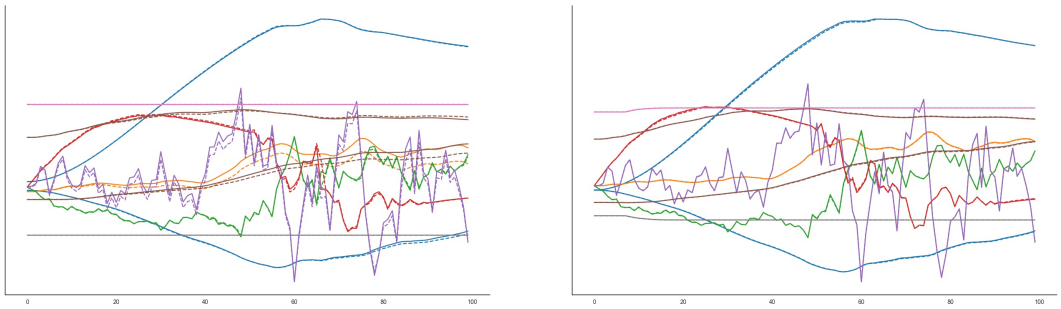
- *Pusher*: A 3-DOF arm trying to move a cylinder on a plane to a specific location;
- *Striker*: A 7-DOF arm that has to strike a ball on a table in order to make the ball move into a goal that is out of reach for the robot;
- *ErgoReacher*: A 4-DOF arm that has to reach a goal spot with its end effector.

While we added backlash to only one joint of the Pusher, to test the limits of our artificial reality gap, we added backlash to three joints of the Striker. We also compare our proposed method with different baselines:

- *Expert policy*: policy trained directly in the target environment;
- *Source policy*: transferring a policy trained in source environment without any adaption;
- *Forward model policy*: a forward dynamic model ψ is trained using an LSTM and data collected from the target domain then a policy trained using only this model (without insight from the source domain);
- *Transfer policy*: the policy trained using NAS.

1.1.4.2 Trajectory following

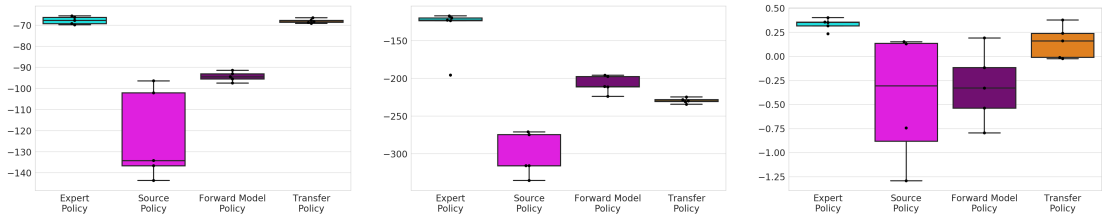
We evaluated the two models learned ϕ and ψ on a 100-step trajectory rollout on the Pusher (see Figure 1.4). The forward model ψ is trained without access to the source domain and is making predictions in an open-loop. While this model is accurate at the beginning of the rollout, its small error compounds over time. We will see later that this makes it hard to learn policies that will achieve a high reward. On the other hand, the model ϕ is grounded using the source environment and only



(a) Forward model rollout **(b)** Source environment + model rollout

Figure 1.4: Starting from the same initial state, a fixed sequence of actions is executed in D_t and compared with the trajectory derived from a forward model and our method. The solid line depicts the true trajectory in D_t whereas the dotted line represents the trajectory using our corrected model, 1.4a is just an LSTM and 1.4b is an LSTM and model Ψ

needs to correct the simulator predictions, so the difference between the trajectory in the real environment is barely noticeable. It shows that correcting trajectories from the source environment provide an efficient way to model the target environment.



(a) Pusher **(b)** Striker **(c)** ErgoReacher

Figure 1.5: Comparison of the different methods described when deployed on the target environment, for the Pusher (1.5a) and the Striker (1.5b).

1.1.4.3 Simulated Environments - Sim to Sim transfer

We tested our method on the simulated environments previously introduced. The number of trajectories used to train the models varied from 250 for the Pusher over 1k for the ErgoReacher to 5k for the Striker. Policies are trained for 2M frames and evaluated over 100 episodes, averaged across 5 seeds, results are given in Figure 1.5.

Our experiments show that in all our environments the policy learned using NAS improve significantly over the source policy, and even reach expert performance on the Pusher. Though it seems that the forward model policy is doing better on the Striker, this is just a consequence of reward

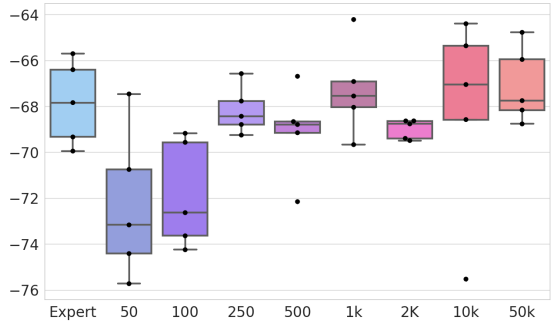


Figure 1.6: We compare the results of our method when the number of trajectories used to train the model φ varies on the Pusher

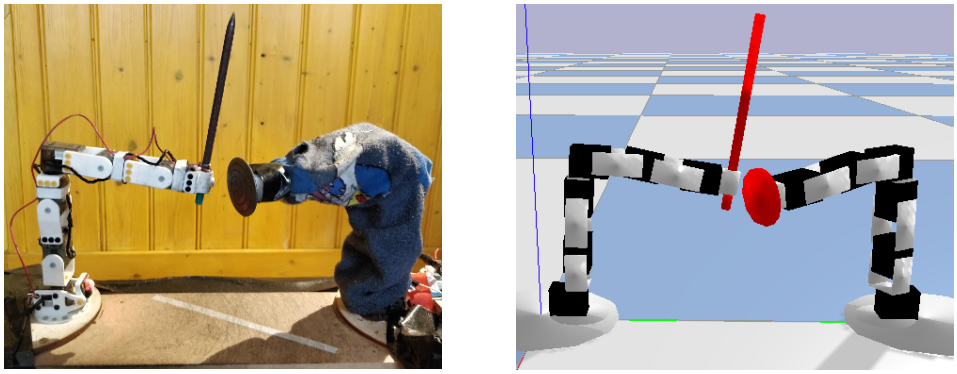
hacking; the agent learns a single movement that pushes away the ball without considering the target position. In contrast, the policy learned using NAS aims for the right direction but does not push it hard enough to make it all the way to the target. This happens when, following a random policy in the target domain, we do not record enough interaction between the robot and the ball to model this behavior correctly. An expert policy (e.g. given by human demonstration) could help alleviate this issue and make sure that the relevant parts of the state action space are covered when collecting the data. Altogether it highlights the fact that we cannot hope to learn a good transfer policy for states where there exists both a significant discrepancy between the source and target environments and insufficient data to properly train the LSTM.

We also varied the number of trajectories used to train the model in NAS to see how it influences the final performance of the transferred policy, see Figure 1.6. When more than 100 trajectories are used, the difference with the expert policy is not visually noticeable and the difference in reward is only due to variance in the evaluation and policy learning. This is in contrast to the 3-4k trajectories required by the expert policy during training to reach its final performance.

1.1.4.4 Physical Robot Experiments Overview

We use the existing Poppy Ergo Jr robot arm in a novel task called *"ErgoShield"*, a low-cost physical variant of the OpenAI Gym *"Reacher"* task. Each robot arm has 6 DOF: with respect to the robot's heading 1 perpendicular joint at the base, followed by 2 in parallel, 1 perpendicular, and two more in parallel. All joints can rotate from -90 degrees to +90 degrees from their resting position.

For this task we fixed two of these robots onto a wooden plank, facing each other at a distance that left the two tips 5mm apart in resting position. One robot is holding a "sword" (standard BIC 19.5cm wooden pencil, sticking out of the pen holder end effector by about 2cm at the bottom) and



(a) Physical Robots **(b)** Simulation

Figure 1.7: The *ErgoShield* environment in reality and simulation. The attacker (sword, left side) has to hit the randomly-moving defender (shield, right side) as often as possible in 10s. In the left image the defender robot is covered in a sock to mitigate the attacker getting stuck too often. The joint angles were compensated for backlash in the left image to highlight similarities.

the other is holding a shield (2.35cm in radius and 0.59cm deep in the center*). The shield robot ("defender") moves every 2.5s to a random position in which the shield is in reach of the opponent. The sword robot ("attacker") is the only robot directly controllable. Each episode lasts 10s and the attacker's goal is to touch the shield as often as possible. Every time a hit is detected, the robots reset to a resting position (attacker) and different random position (defender). The random sampling ranges in degrees on each joint of the shield robot are [45, 15, 15, 22.5, 15, 15] respectively centered around the resting position. The setup is depicted in Figure 1.7. This environment is accompanied by a simulation in PyBullet†‡.

The environment can be controlled at 100Hz by sending a 6-element vector in range [-1,1] corresponding to the desired motor position. The environment is observed at 100Hz as a 24-element vector consisting of: attacker joint angles, attacker joint velocities, defender joint angles, defender joint velocities. In the physical robots, the hit detection is implemented by coating both sword and shield in conductive paint, to which a MakeyMakey§ is attached.

In this task, the noise and non-stationarities can stem from:

- the gear backlash on the Dynamixel XL-320 servos;

*A 3D-printable version is available at https://www.tinkercad.com/things/8UXdY4a5xdJ-ergo-jr-shield#/

†https://pybullet.org/wordpress/

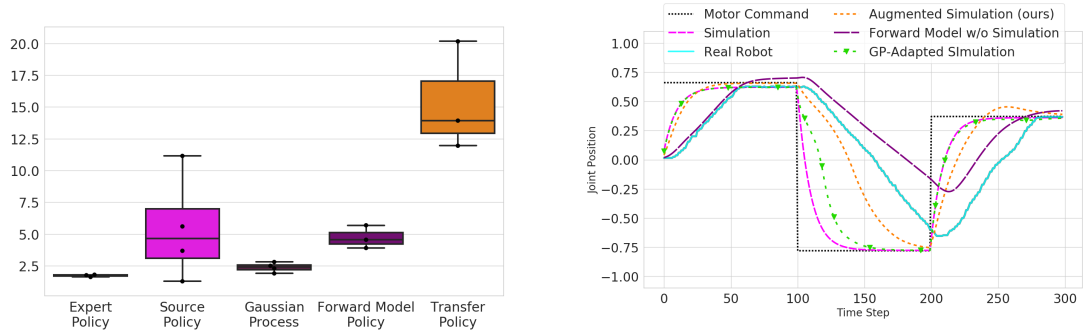
‡The simulated environment is also available on GitHub at https://github.com/fgolemo/gym-ergojr

§https://makeymakey.com/

- the body of the robot being 3D-printed out of thin PLA plastic;
- the body parts being mounted to the servos via plastic rivets;
- overheating of the motors and wear of the body parts; and/or
- the electro-conductive paint thinning out over time.

For the offline data collection, a single robot with an empty pen holder end-effector was instructed to move to 3 random positions for one seconds each (corresponding to an episode length of 300 frames). We collected 500 such episodes equivalent to roughly half an hour of robot movement including resetting between episodes.

1.1.4.5 Results on Sim-to-Real Transfer



(a) Method Comparison (Real Robot)

(b) Example Single Joint Position and Estimate over Time

Figure 1.8: Results of different simulation to real transfer approaches. Left: comparison of average rewards of 20 rollouts of 3 policies per approach. Right: comparison of single joint behavior when receiving a target position (violet dotted) in simulation (green dashed), on the real robot (blue solid), and estimates from the forward model (red, dashed) and our method (yellow, dotted)

We followed the same experimental paradigm as in the sim-to-sim experiments in 1.1.4.1: 3 expert policies were trained directly on the real robot, 3 policies were trained in simulation and were evaluated on the real robot, 3 policies were trained using our method, and 3 policies were trained with only the forward dynamics model. In addition to the forward model and neural-augmented simulation, we also trained an additional baseline, a Gaussian Process regression (GPR) model. It was trained identically to the NAS, to predict the state discrepancy from the last-known real state, the next simulated state, and the action. The GPR underwent a hyperparameter/kernel search before

deployment in policy-learning. Among the tested kernels*, *constant*, *RBF*, *constant* $*$ *RBF*, *ESS*, and *constant* $*$ *ESS*, we found the *constant* $*$ *RBF* kernel to have the lowest prediction error on a small training set of 100 random samples from the dataset of real robot trajectories. Since the complexity of the GPR algorithm is in $\mathcal{O}(n^3)$ (Rasmussen, 2006), we weren't able to fit out model to the entire dataset. More than 2000 samples made our lab computers run out of memory. We found, however, diminishing returns in terms of accuracy gain when improving the dataset sample size, as can be seen in Table 1.1. To augment the simulation, we used the 2000-sample model.

Samples	MSE on Test Set
10	0.0215
100	0.0152
1000	0.0144
2000	0.0140

Table 1.1: Performance of the Gaussian Process Regression model on a held-out test set, as measured by mean squared error with respect to how many samples were used to train the model. The prediction accuracy seems to increase asymptotically with increased sample number.

All policies were trained with the same PPO hyperparameters, save for the random seed. The hyperparameters are listed in Table 1.2.

The evaluation results are displayed in Figure 1.8a and a video is available at https://youtu.be/nw9YTbCEYH8. The policies trained directly on the real robot performed significantly worse than all the other policies. The main reasons for this are (a) the hit detection is not perfect (as in simulation) and since not every hit gets detected the reward is sparser†. And (b) since the attacker robot frequently gets their sword stuck in the opponent, in themselves, or in the environment, exploration is not as easy as in simulation.

We did not find a significant difference in the performance of the simulation and forward dynamics policies. However, our approach (the "Transfer policy") yielded a significantly better results than any others.

Figure 1.8b shows for a single joint how the different approaches estimate the joint position under a given action. The simulation approaches the target position asymptotically while the real robot

*RBF - Radial-basis function, ESS - Exp-Sine-Squared function, see http://scikit-learn.org/stable/modules/gaussian_process.html#basic-kernels for the implementation details.

†There are no false positives, but we estimate that $10 - 15\%$ hits aren't recognized.

Parameter	Value	Parameter	Value
entropy-coef	0	num-stack	1
gamma	0.99	num-steps	2048
lr	3e-4	ppo-epoch	10
num-frames	1000000	seed	[0,1,2,3]
num-mini-batch	32	tau	0.95
num-processes	4	use-gae	True

Table 1.2: PPO-Hyperparameters used for training all the policies on the real robot

approaches the same value linearly. The GPR trajectories did not predict a noticeable deviation from the raw simulator trajectory. This is likely due to the fact that the GPR model does not retain memory of the residual from the previous timestep. It is worth noting that even though the forward model can estimate the recorded dataset very well, policies trained using only this model and no simulation tend to perform badly. This is likely due to the forward model overfitting to the training data and not generalizing to other settings. This is a crucial feature of our method: by augmenting the simulator we are able to utilize the same learned augmentation to different tasks.

Table 1.3 displays the exact quantitative differences between the expert policy (the policy rolled out on the real robot) and (a) the source policy (same policy in simulation), (b) the forward model policy (the policy rolled out through the LSTM), (c) the transferred model policy (our approach for modifying the simulator output), and (d) the Gaussian Process regression model (using a GPR instead of an LSTM to augment the simulator).

The point-wise difference in (a) indicates the expected significant deviations between simulation and real robot. The low deviations in (b) are specifically what that model was trained for and are therefore near-zero. In practice however, this leads to overfitting, i.e. the so-trained model doesn't perform well on policies for which this model has not been exposed to any trajectories (which is evident from the performance in Figure 1.8a). The differences in (c) show that the modified simulator is closer to the real robot trajectory. In combination with the final performance in Figure 1.8a, we can infer that our model does not overfit as the forward model does because it is harnessing the simulator to generalize to previously unseen trajectories.

	1st Quartile	Mean	3rd Quartile
(a) Expert-Source	0.027	0.120	0.172
(b) Expert-Forward Model	0.002	0.008	0.009
(c) Expert-Transferred M.	0.015	0.063	0.078
(d) Expert-GPR Model	0.022	0.107	0.144

Table 1.3: Quantitative analysis of trajectory differences on the ErgoShield task, calculate by the sums of squared differences at each point in the trajectory over 1000 trajectories.

1.1.5 Conclusion

Currently, deep reinforcement learning algorithms are limited in their application to real world scenarios by their high sample complexity and their lack of guarantees when a partially trained policy is deployed. Transfer learning from simulation-trained agents offers a way to solve both these issues and enjoy more flexibility. To that end, we introduced a new method for learning a model that can be used to augment a robotic simulator and demonstrated the performance of this approach as well as its sample efficiency with respect to real robot data.

Provided the same robot is used, the model only has to be learned once and does not require policy-specific fine-tuning, making this method appropriate for multi-task robotic applications.

In the future, we would like to extend this approach to more extensive tasks and different robotic platforms to evaluate its generality, since working purely in simulation leaves some noise that occurs in real robots unmodeled. We would also like to move to image-based observations because our current action/observation spaces are low-dimensional but have a very high frequency (50Hz in sim, 100Hz on real robot). Since our approach is already neural network-based and neural networks are known to scale well to high dimensionality, this addition should be straightforward.

Another interesting path to investigate would be to combine more intelligent exploration methods for collecting the original dataset. If the initial exploration is guided by intrinsic motivation or count-based exploration it might further improve the sample-efficiency and reduce the amount of random movements that need to be recorded in the real robot.

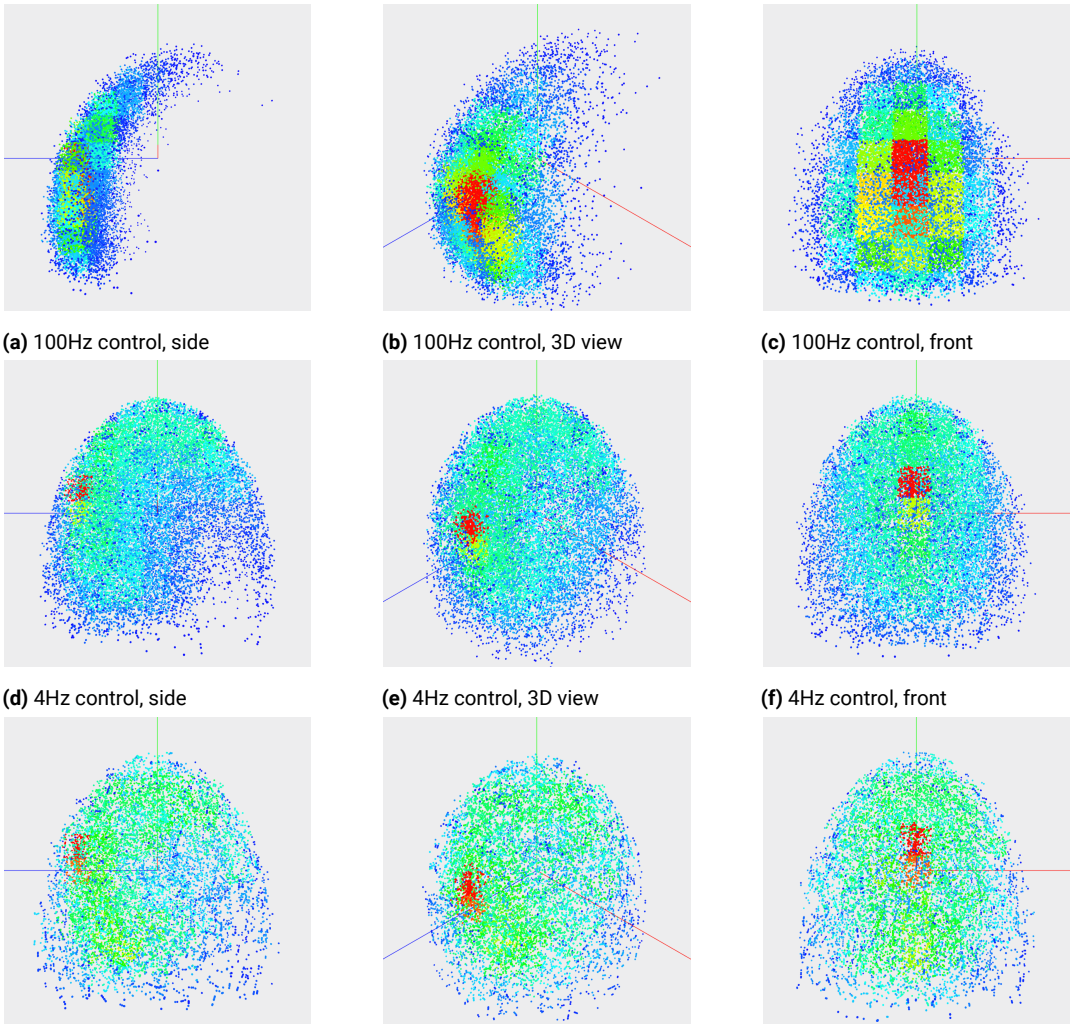
1.2 Curiosity: A State Space Odyssey

Let's go explore. Rolf, Steil, and Gienger (2010) demonstrated how different exploration methods contribute to the state space coverage in an agent. This is especially crucial in the case of learning robot dynamics. In this chapter we give a brief overview over how a single experiment parameter can heavily influence state space coverage, and lay the groundwork for how in the future goal babbling could be incorporated into the NAS method.

1.2.1 Control Frequency

When experimenting with any robot, one has to set the control frequency at which new control signals are being sent to the robot. In case of the Poppy Ergo Jr. robot we can do this up to 100 times per second. Every time a new action gets sent, the previous action is interrupted. In our initial experiments, we allowed the robot to be controlled at 100Hz and therefore, when gathering the dataset on the real robot, we instructed the robot to move to uniformly random positions at 100Hz. In order to analyze the effective state space coverage of this approach, we created an interactive 3D heatmap of the end effector positions. The code for this visualization is available at https://github.com/fgolemo/sim2real-densityplot and the results for the naive case of 100Hz exploration can be found in Figure 1.9a-1.9c. We can observe that the high control frequency leads to an exploration only around the resting position of the end effector. The back and sides of the robot weren't reached at all. Consequently, we suspect that a model trained on this dataset would fail to correctly predict the simulator/real discrepancy in these areas. For the `ErgoShield` task that we presented in the NAS article, this was sufficient. For the back and sides one could imagine either a simple 360°version of the `Reacher` task, or putting the robot on a randomly rotating platform in the ErgoShield task. At the time of writing, this evaluation is still in development.

The reason for the low state diversity in the naive case is that the high frequency of commands averages out to roughly zero around the resting position before the robot can accelerate into any specific direction. The straight-forward solution to this is lowering the control frequency. We've experimented with sending an action at 4Hz and 1Hz. The results of the state coverage of the robot's tip can be seen in Figure 1.9d-1.9f and 1.9g-1.9i, respectively. In case of the 4Hz control, we noticed an overall significantly more even distribution but a large blind spot right behind the robot's base. This is resolved in case of the 1Hz control.



(a) 100Hz control, side **(b)** 100Hz control, 3D view **(c)** 100Hz control, front

(d) 4Hz control, side **(e)** 4Hz control, 3D view **(f)** 4Hz control, front

(g) 1Hz control, side **(h)** 1Hz control, 3D view **(i)** 1Hz control, front

Figure 1.9: We are comparing how different control frequencies affect state space coverage of the end effector position. The color represents a heatmap of density, where bright red is very dense, yellow is medium dense and dark blue is low density. The data for each row was created by sending random actions to the Poppy Ergo Jr. 1000 times for roughly 3s and logging the end effector position at each point in time, before resetting the robot to its resting position. The visualization was constructed by randomly sampling 20k random end effector positions from each of the 3 datasets (100Hz, 4Hz, and 1Hz control respectively). These points were normalized to be in $[-1, 1]$ along all axes and a heatmap was created by grouping the points into a $10 * 10 * 10$ grid, measuring the relative count of points in each grid cell.

1.2.2 Better Exploration

While this approach works for a simply better state space exploration, especially low-cost robots behave very differently when the control frequency changes. In the case of the Ergo Jr., the signifi-

cant backlash (indicated in an acceleration delay, which can be seen in the cyan plot in Figure 1.8b) is emphasized in high-speed control. Therefore, lowering the control frequency only aids state space exploration unless the task environment is controlled at the same frequency. In other words, the dynamics model learned by the NAS approach only benefits from this in cases where the robot doesn't have to move rapidly.

As a better solution to this problem, we propose random or active goal babbling as initial exploration policy. These approaches were introduces in Rolf et al. (2010) and Baranes and Oudeyer (2013) respectively. We used the "Explauto" library* to implement this on the real robot. These experiments are currently still ongoing.

*https://github.com/flowersteam/explauto

Did you know that the first Matrix was designed to be a perfect human world? [...] It was a disaster. No one would accept the program.

The Wachowski Brothers (from "The Matrix")

2 Robo Fiction

But first, we need a simulation. In the last chapter we assumed that we have, not just the real robot, but also a good simulation environment in place. But what if we didn't? In this chapter we dive into **developing the simulations** that make sim2real transfer possible in the first place.

This chapter contains 4 sections. We start out in section 2.1 by bridging the gap from the previous chapter and looking at the simulations that we mentioned there in more detail, including how they came into being. This section also contains additional simulations that were developed over the course of this doctoral program.

Section 2.2 introduces Duckietown, a new robotic and simulated environment for which there isn't a sim2real implementation yet. But since the project was also developed to be cheap and beginner-friendly, we thought it would be a great opportunity to open this problem to the machine learning community and host a sim2real challenge at NIPS 2018, similar to RoboCup (Kitano, Asada, Kuniyoshi, Noda, & Osawa, 1997).

While there are many robot simulations, we realized that for training a robot on human interaction there weren't any robo-centric datasets. Section 2.3 details our attempt at creating such a training corpus. The task made it hard to collect enough data to end-to-end-train contemporary neural

networks, nevertheless, in that section we propose a dataset that we believe is well-suited for testing future human-robot interaction methods in a harsh, natural, noisy environment.

All these simulations already contain a robot, but are very limited in terms of environmental complexity. The final section (2.4) removes the robot as central focus but extends the environments to richly decorated apartments and even full houses. This opens the problem to different robot bodies and also allows for richer interaction with the environment, such as, through sounds or speech.

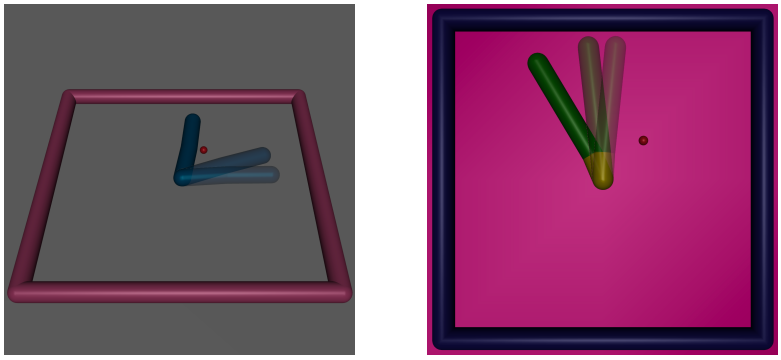
2.1 New Simulators

In this chapter we briefly introduce the four simulation and gym environments that were developed for the Neural-Augmented Simulator. This overview contains three simulator environments, two of which were modified from existing code to support rapid change of dynamics, and one that was created from scratch. The fourth one is a wrapper around the real robot so that the robot can be used for live reinforcement learning following the OpenAI Gym conventions (Brockman et al., 2016). All of these environments come with a full code release that is independently reusable for other projects.

2.1.1 Reacher, Pusher, and a Weird Cheetah

For our sim2sim experiments with the neural-augmented simulator (NAS), we needed to analyze how deviations from the default dynamics behave. We also needed to create a simulation environment that contained backlash on one or more joints and ideally some color change signaling to easily distinguish the different configurations. We wanted to start with the simplest possible robot to validate our idea, which is why we started with the ``Reacher" gym environment (see figure 2.1a). This is a simple arm with 2 joints and two long tube-shaped links, controlled via acceleration values on each joint. The task in this environment is to reach a random point in the arena with the robot's end effector. Most of the robotic gym environments are MuJoCo-based (Todorov et al., 2012), with a robot model loaded from XML. In order to modify the dynamics of the system, the XML file needs to be modified before loading. To that end, we created the ``Reacher2" environment, available at https://github.com/fgolemo/gym-reacher2. In this environment, we intercept the XML loading and allow the user to make modifications to the dynamics. These are written to temporary model files, before being loaded into the actual simulation. The current code allows users to modify the following (the differences are illustrated in figure 2.1):

- Length of both links;
- "Gear vector" (a MuJoCo term), roughly corresponding to the motor gain, i.e. an acceleration multiplier;
- field of view and position of the camera;
- colors of the arena and robot; and
- backlash (enabled or disabled) on specified joints.



(a) Reacher2 default **(b)** Reacher2 modified

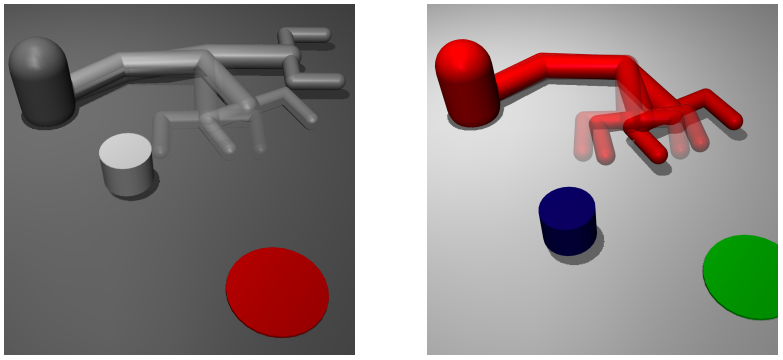
Figure 2.1: Comparison of the Reacher2 environment with default settings (left) and modified settings (right). Both versions used the same starting position and followed the same actions. The right environment has the gain on the inner joint halved and consequentially moves slower.

The default values and usage instructions are provided in the code repository. In addition to these settings, we also provided two variations on this environment; one allowing for the camera image to be included in the observation tuple and one for demonstrating how to implement the NAS state compensation for the Reacher task.

The Reacher environment is, however, very simple. This is why we set out to find more interesting tasks. After some experimentation with the default 7 DoF Pusher and the 7 DoF Striker, we found the Striker sufficiently complex and solvable, but also that the Pusher task was too hard with the given robot complexity. Standard reinforcement learning (RL) techniques were not able to learn a policy for this task without hyperparameter search for the RL algorithm. Since we weren't focusing on the RL algorithm but rather on the sim2real aspect, we adopted the simplified `Pusher3Dof` from Liu, Gupta, Abbeel, and Levine (2018)* for our modified `Pusher3Dof2`. This task is carried out entirely on a horizontal plane, with a 3 joint robot that has to push a cylinder forward on a table into a target area. The modification allows for users to change the following parameters when starting an experiment (figure 2.2 contains an illustration):

- All the settings that are also available in the Reacher2 environment, but for the 3 joints instead of 2;
- density of the robot arm and the pushable object; and
- friction of the table with respect to the object.

*With its corresponding codebase at https://github.com/wyndwarrior/imitation_from_observation



(a) Pusher3Dof2 default **(b)** Pusher3Dof2 modified

Figure 2.2: Comparison of the Pusher3Dof2 environment with default settings (left) and modified settings (right). Both versions started from the same starting position and followed the same actions. The right environment has significantly higher density in the arm and thus higher mass and lower acceleration.

We also experimented with the `HalfCheetah` environment, which ultimately didn't make it into our publication, but received its modified version in `HalfCheetah2`. In the HalfCheetah2 environment, we parameterized the joint gains. Both the Pusher3Dof2 and the HalfCheetah2 received the same additional environments as the Reacher2: the "...Pixel" environment, which stores the camera images as an observation, and the "...Plus" environment, which demonstrates how to use both with the NAS. Both the cheetah and the simplified pusher are available at https://github.com/fgolemo/gym-throwandpush. We also included the `Striker` environment, but without the parameterization - only with a switch to turn backlash on all joints on/off.

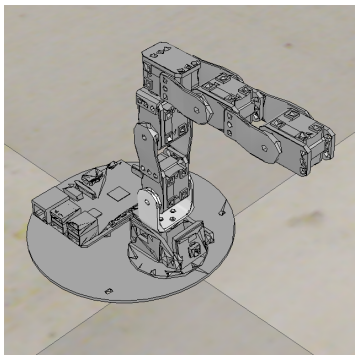
2.1.2 Ergo Jr. Sim & Real

Since the FLOWERS lab at INRIA Bordeaux created the Poppy Ergo Jr. robots, and they constitute a sufficiently complex but low-cost robot platform, we decided to run our sim2real experiments on these. The developer suite for the robots contains a physics-free simulator and a more intricate simulation in the commercial V-REP simulator*, see figure 2.3a. After some experimentation with the V-REP simulation, we found that the simulator didn't support the state correction as proposed in our NAS method. Angular joint velocities would be set to zero and after few iterations, the simulation would become unstable (e.g. links started twitching and passing through the floor). Since this environment wasn't sufficient for our method, we developed a new simulator for the Ergo Jr. robot.

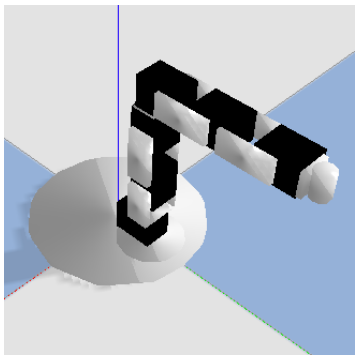
*http://www.coppeliarobotics.com/

2.1.2.1 Simulator

The Bullet physics engine* is open source (as opposed to the proprietary V-REP), and our initial tests showed that the simulation was stable when resetting the environment. Therefore, we rebuilt the Poppy Ergo Jr. from scratch in the Bullet system by measuring and weighing the individual parts of the robot and their position relative to each other. The robot was defined using the URDF definition format that is common in robotics and also supported, for example, by MuJoCo. For the new model, only primitive shapes were used (i.e. boxes and cylinders), in order to speed up the simulation from around 120 in V-REP to around 6000 in Bullet using the same settings (same delta time, same constants). In order to speed up the development of specific gym environments with this simulation, we developed a compositional model import, which allows users to add multiple Ergo Jr. configurations (i.e. end effectors) and multiple Ergo Jr. bases to a scene without copying any code. This is implemented in the URDF macro language XACRO and automatic compilation of the composed scene definitions into URDF files on the first run.



(a) Ergo Jr. VREP



(b) Ergo Jr. Bullet

Figure 2.3: Comparison of the Poppy Ergo Jr. robot in V-REP (left) and Bullet (right). The amount of detail is drastically lower in the Bullet simulation, resulting in faster simulation.

2.1.2.2 Real Robot Interface

For both training a policy directly on the physical robot and for evaluating the simulator-trained policies on the robot, we additionally provide a gym-compatible remote-control interface written in Python, available at: https://github.com/fgolemo/poppy-helpers. By modifying the communication standard by which the robot and operator exchange sensor readings and commands from

*https://pybullet.org/wordpress/

HTTP REST to ZeroMQ*, we are able to send control signals to the robot at around 100Hz and receive readings at the same speed. We measured the latency for each cycle of sending a motor signal and receiving the sensor data to be between 0.5 and 2ms on wired connections and between 1 and 3ms on 2.4Ghz WiFi. The robot control software with the modified message exchange method can be found at https://github.com/fgolemo/pypot and replaces the software package with the same name that was preinstalled on the robot.

*http://zeromq.org/

2.2 The AI Driving Olympics at NIPS 2018

Deep learning and reinforcement learning have had dramatic recent breakthroughs. However, the ability to apply these approaches to control real physically embodied agents remains primitive compared to traditional robotics approaches.

To help bridge this gap, we are announcing the AI Driving Olympics (AI-DO), which will be a live competition at the Neural Information Processing Systems (NIPS) in Dec. 2018. The overall objective of the competition is to evaluate the state of the art of machine learning and artificial intelligence on a physically embodied platform. We are using the Duckietown platform (Paull et al., 2017) since it is a simple and well-specified environment that can be used for autonomous navigation.

The competition comprises five tasks of increasing complexity - from simple lane following to

---

Section 2.2 was published as

Andrea Censi, Liam Paull, Thomas Ackermann, Oscar Beijbom, Berabi Berkai, et al. **The AI Driving Olympics at NIPS 2018**. In Robotics: Science and Systems (RSS) 2018, Workshop Track.

To this work we contributed: Designing the simulator pipeline of the challenge for creating Docker[a] container-enclosed submissions that are automatically evaluated in simulation and are transferable to the real robot without adjustments; code for a control abstraction layer that makes the Duckiebot network-controllable or controllable from other on-board Docker containers without ROS with minimal latency [b], which became the cornerstone of the challenge software infrastructure; implementing both sides of the client-server simulator connection that lets contestants train their policy without direct access to the simulation server and wrapping this as gym-compatible environment; designing and implementing some of the specifications for the tasks like the log format and the map format; designing and implementing part of the containerization infrastructure that encapsulates all submissions and allows for participation across operating systems; and providing the baseline reinforcement learning setup and tutorial that allows participants to train a working lane-following policy in simulation and deploy it onto the real robot. The Montreal Robotarium is not yet complete as of this writing, but we have been tasked to help in the construction of the autonomous evaluation circuit and overseeing the experiments, as well as overseeing the live event at NIPS.

[a] https://www.docker.com/
[b] https://github.com/duckietown/duckietown-slimremote/

managing an autonomous fleet. For each task we will provide tools for competitors to use in the form of simulators, logs, low-cost access to robotic hardware and live "Robotariums" where submissions will be evaluated automatically.

2.2.1 Introduction

Machine Learning methods have shown remarkable success for tasks in both disembodied dataset settings and virtual environments, yet still have to prove themselves in embodied robotic task settings. The requirements for real physical systems are much different: (a) the robot must make actions based on sensor data in realtime, (b) in many cases all computation must take place onboard the platform and resources may be limited, and (c) many physical systems are safety-critical which imposes a necessity for performance guarantees and a good understanding of uncertainty. All of these requirements present a stress on current machine learning methods that are built on Deep Learning (DL), are resource hungry and don't provide performance guarantees "out of the box".

Figure 2.4: Duckietown is a scaled-down autonomous city where the inhabitants are duckies. The platform comprises autonomous vehicles (Duckiebots) and a well-specified environment in which the Duckiebots operate. This platform is used as the basis for the AI Driving Olympics competition which will take place at the NIPS 2018 conference.

"Classical" robotics systems are built as a composition of blocks (perception, estimation, planning, control, etc.), and an understanding of the interplay between these different components is necessary for robots to be able to achieve complex and useful tasks. However, with machine learning methods it is unclear whether these abstractions are necessary and whether it is more effective to learn chains of these components in an "end-to-end" fashion or to separate the components out and learn them in isolation.

In order to address these issues, we have developed a new benchmark for evaluating machine learning methods on physically embodied systems, called the "AI Driving Olympics" or AI-DO. The first AI-DO event will be a live competition at the Neural Information Processing Systems (NIPS) 2018 conference in Montréal, Québec.

A main consideration for casting this research as a competition is that solutions to embodied robotic tasks are notoriously difficult to compare (J. Anderson, Baltes, & Cheng, 2011; Behnke, 2006). As argued by Behnke (2006), competitions are therefore a suitable way of advancing robotics by making solutions comparable and results reproducible. Without a clear benchmark of performance, progress is difficult to measure. To drive home this message, we recall the story of the drunkard who has lost his house keys and then searches for them under the street light. When asked why he searches close to the street light he answers because *it is lighter there* (Kaplan, 2017). Learning from this experience, we aspire to cast this competition in a way that encourages solving the problems that need to be solved rather than the ones we know how to solve.

Furthermore, as argued in Zilly et al. (2017) in the context of computer vision, large scale endeavors with relevant real-world applications have the potential to push the envelope of what is possible further in a process of *productive failure* (Kapur, 2008). *Productive failure* is a concept pioneered by Kapur which emphasizes the importance of productive struggle in the context of learning.

2.2.1.1 Novelty and related work

We call this competition the "AI Driving Olympics" (AI-DO) because there will be a set of different trials that correspond to progressively more sophisticated behaviors for cars. These vary in complexity, from the reactive task of lane following to more complex and "cognitive" behaviors outlined in section 2.2.3.

The competition will be live at the Neural Information Processing Systems (NIPS) conference, but participants will not need to be physically present—they will just need to send their source code.

Competition	DARPA	KITTI	Robocup	RL comp.	HuroCup	AI-DO
Accessibility	-	✓	✓	✓	✓	✓
Diverse metrics	✓	-	✓	-	✓	✓
Modularity	-	-	-	-	-	✓
Resource constraints	✓	-	✓	-	✓	✓
Simulation/Realism	✓	✓	✓	-	✓	✓
ML compatibility	-	✓	-	✓	-	✓
Embodiment	✓	-	✓	-	✓	✓
Teaching	-	-	-	-	-	✓

Table 2.1: *Competition and Benchmark Comparison*. DARPA: Mahelona et al., 2007, KITTI: Geiger, Lenz, and Urtasun, 2012, Robocup: Kitano, Asada, Kuniyoshi, Noda, and Osawa, 1997, RL comp.: Kidziński et al., 2018, HuroCup: Baltes, Tu, Sadeghnejad, and Anderson, 2017

There will be qualifying rounds in simulation, similar to the recent DARPA Robotics Challenge (Iagnemma & Overholt, 2015), and we will make the use of "robotariums," (Pickem et al., 2017), facilities that allow remote experimentation in a reproducible setting more closely described in section 2.2.2.

Many competitions exist in the robotics field. One example is the long-running annual Robocup (Kitano et al., 1997), originally thought for robot soccer (wheeled, quadruped, and biped), and later extended to other tasks (search and rescue, home assistance, etc.). Other impactful competitions are the DARPA Challenges, such as the DARPA Grand Challenges (Mahelona et al., 2007) in 2007-8 that rekindled the field of self-driving cars, and the recent DARPA Robotics Challenge for humanoid robots (Iagnemma & Overholt, 2015).

In machine learning in general, and at NIPS in particular, there exist no competitions that involve physical robots. Yet, the interactive, embodied setting is thought to be an essential scenario to study intelligence (Pfeifer & Scheier, 2001). The application to robotics is often stated as a motivation in recent machine learning literature (e.g., Chaplot, Parisotto, and Salakhutdinov (2018), Higgins et al. (2017) among many others). However, the vast majority of these works only report on results *in simulation* (Kidziński et al., 2018) and on *very simple* (usually grid-like) environments.

To highlight what makes robotics and this competition unique, we list essential characteristics comparing existing competitions to the upcoming AI-DO as outlined in Tab. 2.1.

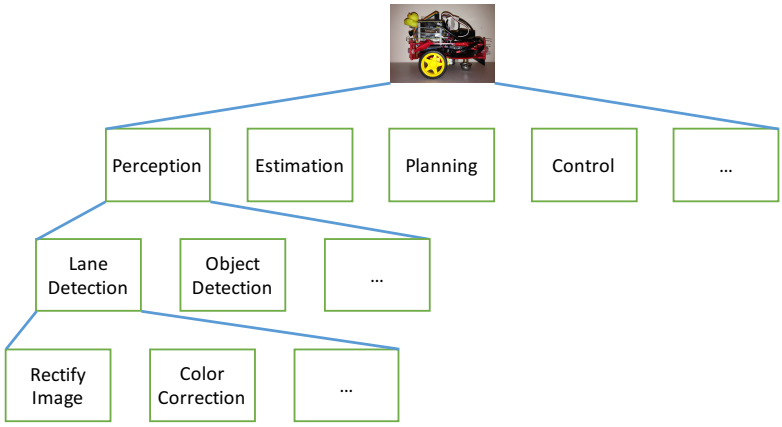


Figure 2.5: Modularity - The full robotics problem is achieved through a hierarchical composition of blocks. Which are the right components and what is the right level of hierarchy to apply machine learning?

- **Accessibility:** No upfront costs other than the option of assembling a Duckiebot are required. Conceptually, classes teaching robotics and machine learning in the Duckietown setting will be made available online as described in section 2.2.6.
- **Resource constraints:** In robotics constraints on power, computation, memory and actuator constraints play a vital role.
- **Modularity:** More often than not, robotic pipelines can be decomposed into several modules (see Fig. 2.5).
- **Simulation/Realism:** Duckietowns will be made available both in simulation and reality posing interesting questions about the relationship to each other.
- **ML compatibility:** Additionally past data and ongoing data from cars in Duckietown will be made available to allow for training of ML algorithms.
- **Embodiment:** As any real robot, closed-loop and real-time interactive tasks await the participants of the competition.
- **Diverse metrics:** In most real-world settings, not one single number determines performance on a task. Similarly AI-DO employs multiple diverse performance metrics simultaneously.

2.2.2 The Platform

We make available a number of different resources to competitors to help them build, test, and finally evaluate their algorithms:

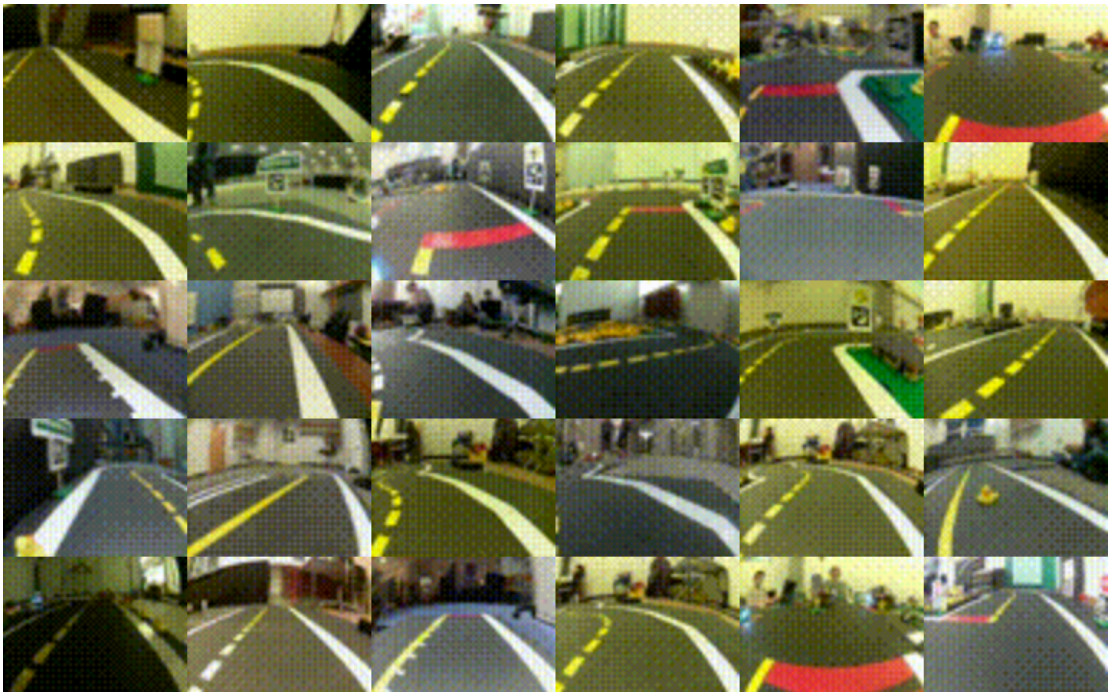


Figure 2.6: Currently 17 hours of logs are available from 38 different robots with more being continuously added.

1. **The physical Duckietown platform (Paull et al., 2017; Tani et al., 2016)** (Fig. 2.4): miniature vision-based vehicles and cities in which the vehicles drive. This is an affordable setup (~$200/robot), with rigorously defined hardware and environment specifications (e.g., the appearance specification*, which enables repeatable experimentation in the online documentation).

2. **Data** (Fig 2.6): Access to many hours of logs from previous use of the platform in diverse environments (a database that will grow as the competition unfolds).

3. **A cloud simulation and training environment**: for testing in simulation before trying on the physical fleet of robots.

*http://purl.org/dth/duckietown-specs

4. **"Robotariums"**: remotely accessible, completely autonomous environments, continuously running the Duckietown platform. Robotariums will allow participants to see their code running in controlled and reproducible conditions, and obtain performance metrics on the tasks.

2.2.2.1 The Development Pipeline

Given the availability and relative affordability of the platform, we expect most participants to choose to build their own Duckietown and Duckiebots and develop their algorithms on the physical platform. Alternatively, we provide a realistic cloud simulation environment. The cloud simulation also serves as a selection mechanism to access a robotarium. The robotariums enable reproducible testing in controlled conditions. Moreover, they will produce the scores for the tasks by means of a network of street-level image sensors. The robotarium scores are the official scores for the leader board and they are used for the final selection of which code will be run at the live competition at NIPS. The participants will not need to be physically at NIPS — they can participate remotely by submitting a Docker container, which will be run for them following standardized procedures.

2.2.2.2 The physical Duckietown platform

The physical Duckietown platform comprises autonomous vehicles (*Duckiebots*) and a customizable model urban environment (*Duckietown*).

The Robot Duckiebots are equipped with only one *sensor*: a front-viewing camera with 160° fish-eye lens, streaming 640×480 resolution images reliably at 30 fps. *Actuation* is provided through two DC motors that independently drive the front wheels (differential drive configuration), while the rear end of the Duckiebot is equipped with a passive omnidirectional wheel. A minimum radius of curvature constraint is imposed, at software level, to simulate more car-like dynamics. All the *computation* is done onboard on a Raspberry Pi 3 B+ computer, equipped with a quad-core 1.4 GHz, 64 bit CPU and 1 GB of RAM. We might support other configurations for the purposes of deploying neural networks onto the robots. *Power* is provided by a 10 Ah battery, which guarantees several hours of operation.

The Environment Duckietowns are modular, structured environments built on two layers: the *road* and the *signal* layers (Fig. 2.7). There are six well defined *road segments*: straight, left and

right 90 deg turns, 3-way intersection, 4-way intersection, and empty tile. Each is built on individual tiles, and their interlocking enables customization of city sizes and topographies. The appearance specifications detail the color and size of the lines as well as the geometry of the roads. The signal layer comprises of street signs and traffic lights.

In the baseline implementation, *street signs* are AprilTags (in union with typical road sign symbols) that enable global localization and interpretation of intersection topologies by Duckiebots. The appearance specifications detail their size, height and positioning in the city. *Traffic lights* provide a centralized solution for intersection coordination, encoding signals in different LED blinking frequencies.

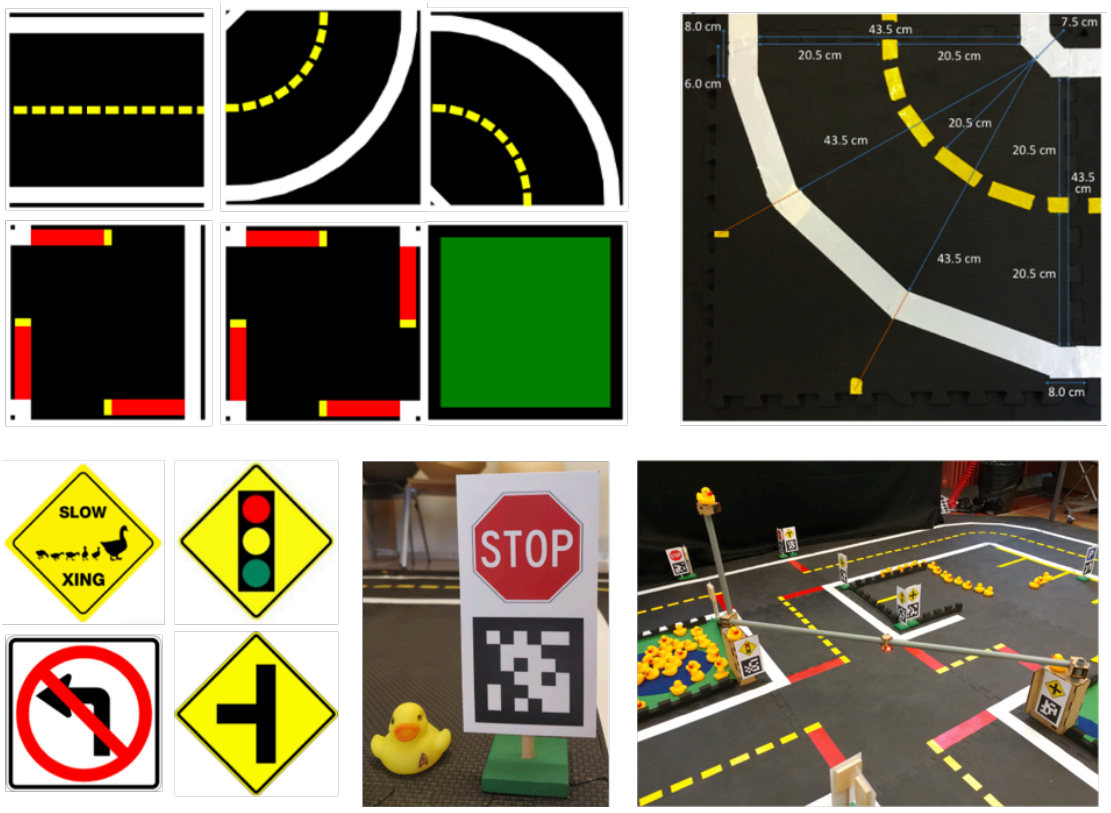


Figure 2.7: The Duckietown environment is rigorously defined at road and signal level. When the appearance specifications are met, Duckiebots are guaranteed to navigate cities of any conforming topology.

2.2.3 Tasks

Many recent works in deep (reinforcement) learning cite robotics as a potential application domain (Chaplot et al., 2018; Higgins et al., 2017). However, comparatively few actually demonstrate results on physical agents. This competition is an opportunity to properly benchmark the current state of the art of these methods as applied to a real robotics system.

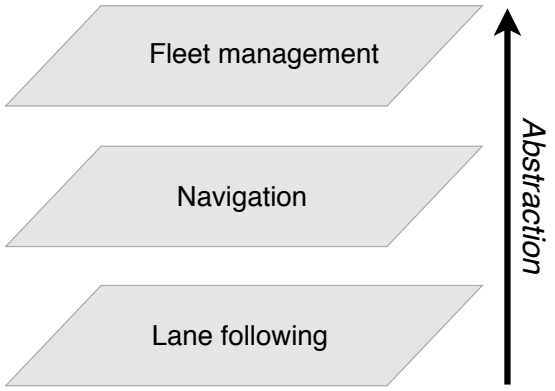


Figure 2.8: The tasks are designed to be increasingly complex, and consequently investigate the autonomous mobility on demand problem at increasing levels of abstraction.

Our experience thus far indicates that many of the inherent assumptions made in the machine learning community may not be valid on real-time physically embodied systems. Additionally, considerations related to resource consumption, latency, and system engineering are rarely considered in the machine learning domain but are crucially important for fielding real robots. We hope this competition can be used to benchmark the state of the art as it pertains to real physical systems and, in the process, spawn a more meaningful discussion about what is necessary to move the field forward.

The best possible outcome is that a larger proportion of the machine learning community redirects its efforts towards real physical agents acting in the real world, and helps to address the unique characteristics of the problem. The guaranteed impact is that we can establish a baseline for where the state of the art really is in this domain.

We propose a sequence of five challenges, in increasing order of difficulty. We briefly discuss the goal of each challenge; later, we will discuss the metrics in detail.

LF **Lane following on a closed course**, with static obstacles (cones and parked vehicles). The robot will be placed on a conforming closed track (with no intersections) and should follow the track on the right-hand lane.

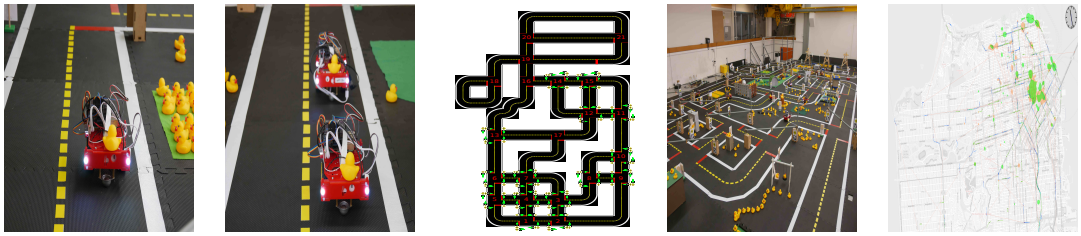


Table 2.2: *Overview of tasks in pictures. From left to right: Lane following, lane following with dynamic vehicles, navigation, fleet management and autonomous mobility on demand.*

LFV **Lane following** on a continuous closed course, with static obstacles **plus dynamic vehicles** controlled "by the matrix" sharing the road. Now the agent needs to deal with an environment populated by other intelligent agents.

NAVV **Navigation plus vehicles** controlled by the matrix. Requires that the robot implement a coordination protocol for navigating the intersections.

FM **Fleet management**: This scenario is meant to resemble a taxi-service. Customer request to go from location "A" to location "B" arrive sequentially and need to be served in an intelligent manner. Participants are asked to submit a central dispatcher that submits navigation tasks to Duckiebots to best serve customer requests.

AMOD **Autonomous mobility on demand**: in addition to dynamic navigation, participants must implement a central dispatcher that provides goals to the individual robots. Points are given for the number of customers served.

2.2.4 Metrics and Judging

Each challenge will have specific objective metrics to quantify success. The success metrics will be based on measurable quantities such as speed, timing and automated detection of breaches of rules of the road. No human judges are necessary.

There are going to be three classes of metrics:

1. **Traffic laws compliance metrics**, to penalize the illegal behavior (e.g. not respecting the right of way);
2. **Performance metrics**, such as the average speed, to penalize inefficient solutions;
3. **Comfort metrics**, to penalize unnatural solutions that would not be comfortable to a passenger.

We will keep these metrics separate, and evaluate the solutions on a partial order, rather than creating a total order by collapsing the metrics in a single reward function to optimize. We will impose minimum objectives (minimum speed, maximum violation) to avoid degenerate solutions (e.g. a robot that does not move is very safe, but it does not have an acceptable performance). A more detailed mathematical description of these metrics is provided in the appendix of the published version of this article (Tani et al., 2016).

We anticipate that there will be multiple winners in each competition (e.g. a very conservative solution, a more adventurous one, etc.).

2.2.5 Technical Components

2.2.5.1 Localization

We are building infrastructure in our Robotariums system to automate the localization of the robots. This is essential for two reasons: 1) To evaluate the score of the agents, and 2) To robustly automate the process of resetting the agents at the their initial configurations in preparation for the evaluation of the next entry.

The approach for automatic localization is based on an system of cameras mounted on traffic lights and other elevated city structures, which are designed to locate the robots and report on their locations (Fig. 2.9).

Additional solutions might be used to increase overall robustness (e.g., ceiling mounted cameras).

2.2.5.2 Containerization

In order to lower the barrier of entry for participants and minimize the amount of code refinements between platforms (simulators, robotarium, real robot), we are developing a state-of-the-art containerization infrastructure, which is also compatible with the cloud interface.

2.2.5.3 Simulation

We are developing a suite of simulators that are available to the competitors for use in development. These simulators scale in fidelity with the complexity of the tasks:

- we have a very lightweight "machine-learning friendly" simulator with minimal dependencies and a low-level API based on OpenGL (shown in Fig. 2.11) that is used for lane following tasks;
- for navigation tasks, we provide a higher fidelity simulator based on Unity;

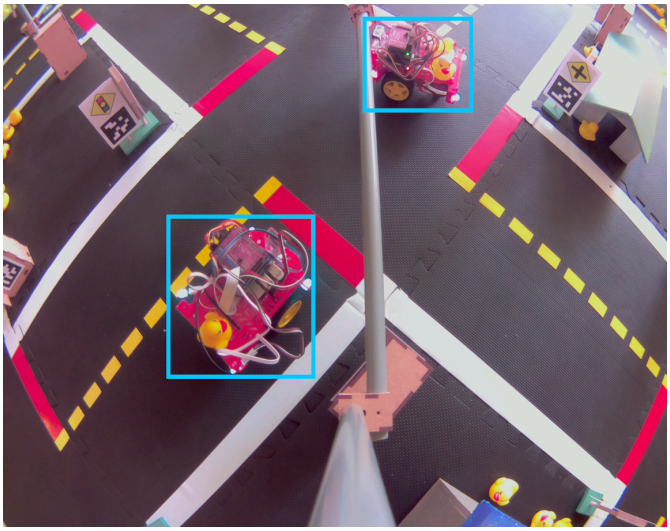


Figure 2.9: The overhead cameras are used to provide ground truth localization of the robots inside the Robotarium environments.

- for the AMOD task, we provide a fleet-level simulator.

2.2.5.4 Baselines

We are providing baseline "strawman" solutions, based both on "classical" and learning-based approaches. These solutions are fully functional (they will get a score if submitted) and contain all the necessary components, but should be easily beatable. Additionally, the baseline solutions will be modular, i.e., they will contain a composition of containers, where competitors can choose to

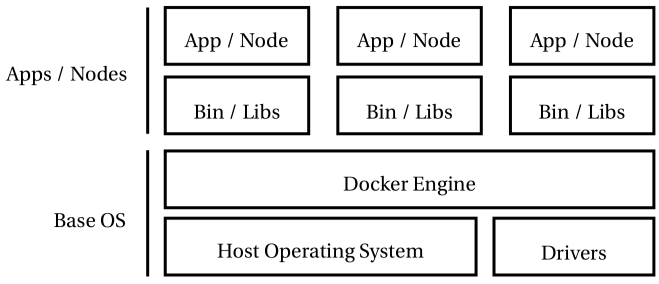


Figure 2.10: Our Docker containerization system acts as an interface between the hardware/simulator and the competitors developed code.

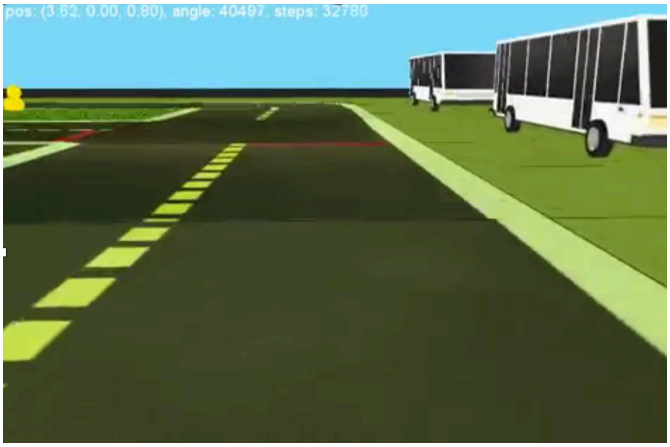


Figure 2.11: Lightweight OpenGL based simulator used for lane following tasks.

replace single containers or combinations (there need not be a one to one mapping between the containers in the baseline solutions and the competitor entries) as long as the well-specified APIs are preserved.

2.2.6 Pedagogical

It is envisioned that there will be two different groups of competitors: (a) independent learners not affiliated with any institution and (b) participants within the framework of a university class. A second version of the globally synchronized class will culminate this year with students submitting entries to the competition. We have provided a full suite of online learning materials in the "Duckiebook" * to support both groups. These materials include slides, theory units, exercises, and demonstration instructions. For more details about the educational components of the projects please see Tani et al. (2016).

2.2.7 Outreach

One of the key design criteria for this project and competition is the low barrier of entry, both in terms of effort and cost. This is very important since we want to encourage participation from all geographical and demographic categories.

To achieve this objective we have:

- designed the containerization framework to allow for rapid development;

*docs.duckietown.org

- striven to keep the hardware components of the platform accessible (mostly off-the-shelf items) and low cost;
- provided access to cloud infrastructure, to ensure a level playing field for all participants.

2.2.8 Conclusion

We are excited to announce a new competition - the AI Driving Olympics, which will take place at the NIPS 2018 conference. The main objective is to evaluate the performance of state-of-the-art machine learning systems in the real physically embodied robot setting. The challenges inherent in deploying robots in the real world are quite different than most other applications where machine learning has had recent breakthroughs. We believe this is an opportunity to inspire the members of the ML community to focus more efforts on the physical embodied scenario, and to provide a benchmark for realistic comparison of algorithms. The platform is designed have a very low barrier for entry, both in terms of cost and in terms of effort, and we therefore hope to attract participants from diverse geographical regions and underrepresented demographics.

2.3 A Multimodal Dataset for Object Model Learning from Natural Human-Robot Interaction

Learning object models in the wild, from natural human interactions, is a key ability for robots to perform general tasks. In this paper we present a robocentric multimodal dataset tackling this important challenge.

Our dataset focuses on interactions where the user teaches new objects to the robot in various ways. It contains synchronized recordings of visual (3 cameras) and audio data which provide a challenging evaluation framework for different tasks.

Additionally, we present an end-to-end system that learns object models from the recorded natural interactions. Our proposed pipeline follows these steps: (a) recognizing the interaction type, (b) detecting the object that the interaction is focusing on, and (c) learning the models from the extracted data. The main novelties are in the steps towards identifying the target object patches. We demonstrate the advantages of combining language and visual features for the interaction recognition and use multiple views to improve the object detection.

Our experimental results show that our dataset is challenging due to occlusions and domain change with respect to typical object learning frameworks. The performance of common out-of-the-box classifiers trained on our data is low. We demonstrate that our algorithm outperforms such baselines.

2.3.1 Introduction

One of the key challenges in service robotics is to achieve an intuitive Human-Robot Interaction (HRI), that feels natural to the user. To achieve this, it is essential that the robot learns models in a

---

Section 2.3 was published as

Pablo Azagra, Florian Golemo, Yoan Mollard, Manuel Lopes, Javier Civera, Ana C Murillo. **A Multimodal Dataset for Object Model Learning from Natural Human-Robot Interaction**. In Intelligent Robots and Systems (IROS) 2017 IEEE/RSJ International Conference.

To this work we contributed: Nearly all of the writing and formatting of this article; all of the tables; some of the diagrams; a large part of the literature research and assessment of related datasets; assistance in the data collection on the robot and data organization in the released dataset; and assistance with the CNN baseline for evaluating the vision pipeline.

Figure 2.12: Object model learning from human-robot interaction. In the proposed pipeline the user interacts with the robot for it to learn new object models. We capture audio and video from this interaction. The data is processed to recognize the interaction type which guides the segmentation of the target object patches in the images. These patches are used to learn new object models.

realistic environment adapted to a particular domain. These models should include objects, scenes, affordances, and capabilities which, in addition, might change over time.

In this work we address the relevant and challenging scenario of a robot learning new object models by interacting with humans in a natural manner. Object models learned from general datasets miss the subtle details of the particular scenes the robot works in. For example a soda can from a specific brand might look the same everywhere, but the appearances of kitchen utensils may vary a lot. In a home deployment scenario existing objects can be modified and new unknown ones can appear. Finally, sensory noise, clutter, and illumination conditions might also change within a domain and cause standard classifiers to fail.

Our contribution along this line of research is twofold: Firstly, we release a partially annotated dataset for object modeling from natural human-robot interactions. Our dataset features robocentric multimodal data from several users and objects, which we found to be novel in this field. Secondly, we propose a full pipeline for acquiring object models from the recorded data (see Fig. 2.12 for an overview). To our knowledge, our proposal is the first one addressing such a multimodal and interactive setting. Our pipeline is designed to address the additional challenges (everyday object segmentation and recognition problems) posed by this natural HRI setup.

Our experimental results underline the challenges of the setting we propose. We need to understand which object the user refers to from the ones available in the scene. Therefore we propose a way to guide different strategies for target object segmentation thanks to a simple initial interaction recognition. Standard object recognition classifiers trained on state-of-the-art object recognition databases exhibit a low performance on our dataset. However, the recognition rate improves when these methods are trained on annotated data from our dataset. This confirms that our data is in a significantly different segment of the domain due to the particularly natural setting of HRI. We evaluate our pipeline and set an initial baseline, presenting promising results about the use of multi-

modal data to enable learning from noisy object patches.

2.3.2 Related work

There are many works in the literature in which the robot interacts directly with the objects in a scene to learn new models. For example, Collet, Berenson, Srinivasa, and Ferguson (2009) created a 3D model of the objects in the scene that a robotic hand has to grasp. Kenney, Buckley, and Brock (2009) proposed to improve object segmentation in cluttered scenarios by manipulating the objects. Additionally there are multiple works which use robotic hands to interact with objects in the scene. For example Iravani, Hall, Beale, Charron, and Hicks (2011) proposed a system where the robot manipulates the objects presented in front of the camera until the model is learned. Krainin, Curless, and Fox (2011) proposed to use a robotic hand to grasp the object and rotate it to obtain different views. Sinapov, Schenck, and Stoytchev (2014) used the robotic hands to interact with plastic jars and obtain multimodal information to learn the content of the jars. These approaches typically need prior information to be able to grasp the objects. Our approach is complementary to these works and focuses on scenarios that require human interaction, e.g. if the object is out the robot's reach the affordances are completely unknown or the grasping capabilities are limited.

For any given robotic platform intended to act in the real world it is necessary to obtain object models. In this sense the approach of Pasquale et al. (2015) is very similar to ours. The authors created a dataset and used CNN-based features and SVM classification for visual recognition. Their training data consists of robocentric images, where a human presents an object in front of the iCub (Metta, Sandini, Vernon, Natale, & Nori, 2008). We improve on these efforts by focusing our dataset and algorithm on realistic multimodality and multiple interaction types. Where these previous approaches solely relied on images we present video, audio, and depth information that can be obtained with common hardware. Furthermore, we extended the interaction to several types which posed the additional challenge of interaction recognition. The dataset presented in Vatakis and Pastra (2016) includes similar multimodal recordings, but it is focused on the psychological reaction of the users.

There are countless datasets for object recognition, e.g., K. Lai, Bo, Ren, and Fox (2011), Singh, Sha, Narayan, Achim, and Abbeel (2014) among the ones containing *RGB-D* images for object recognition (Russakovsky et al., 2015) or using only *RGB* images but containing a enormous variety of objects. However, most of these contain clean images of the objects in a studio, or high resolution pictures of objects in the wild. Whereas such datasets can always be used for offline learning we place our dataset as more realistic by capturing the noise and clutter that would be encountered in

an interactive scenario.

Our work also emphasizes that the point of view is crucial. For example, recognizing a pedestrian from a close-up view of a service robot is very different from performing the same task in the raw video from a distant wide-angle surveillance camera. Datasets like Sung, Ponce, Selman, and Saxena (2011) or Gong, Gonzàlez, Tavares, and Roca (2012) capture human-robot interactions from an external point of view. In the case of mobile robots, using the onboard sensors is more practical than installing sensors everywhere the robot can go.

Additionally, many datasets lack additional sensor data that is easy to find in human-robot interactive scenarios like speech from the user. As detailed in next section, our released dataset is focused not only on capturing the user's expression, but also on capturing the scene information (hands, arms, desk, objects, etc.) related to the object classes being taught to the robot.

2.3.3 Multimodal Human-Robot Interaction Dataset (MHRI)

Our "MHRI" dataset* captures the most common natural interactions to teach object classes to a robot, namely *Point*, *Show*, and *Speak*, from a robocentric perspective. Figure 2.13 shows an example for each considered interaction type (captured from the robot frontal camera):

- *Point*: the user points at an object on the table and announces its name.
- *Show*: the user grabs an object, moves it closer to the robot, and utters its name.
- *Speak*: the user describes where a certain object is in relation to other objects.

(a) *Point* **(b)** *Show* **(c)** *Speak*

Figure 2.13: Examples from the three interaction types in MHRI dataset. The user says, respectively, (a) "This is a box", while pointing at the box, (b) "This is a box", while holding the box, and (c) "The box is next to the chips and has a banana on top."

*Available at http://robots.unizar.es/IGLUdataset/

Table 2.3 summarizes the contents of the dataset. It contains recordings from 10 users and each user performed 10 object interactions of each of the 3 types (*Point*, *Show*, *Speak*), for a total of 300 multimedia short clips. The aforementioned 10 objects per user were picked randomly out of a pool of 22 objects and used by that user for all their recordings. Figure 2.14 illustrates the different sensor modalities of the dataset for different users.

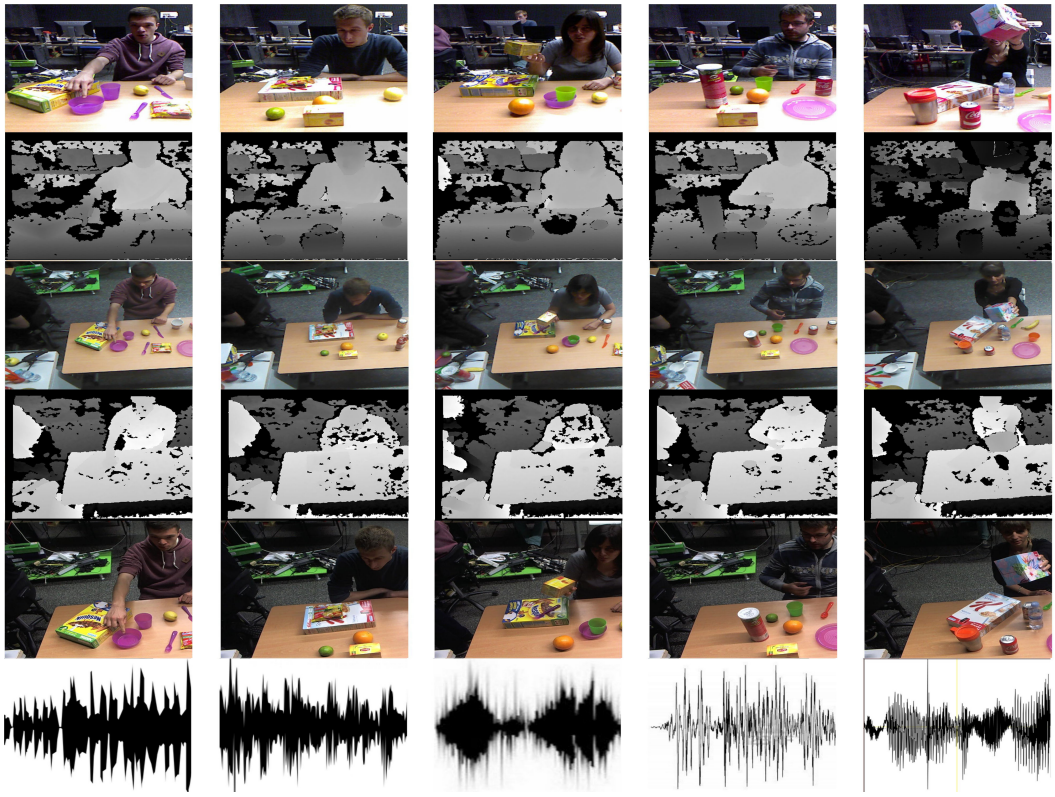


Figure 2.14: Five examples (one user per column) from MHRI dataset. Each row displays a different sensor modality. From top to bottom: *Frontal*-RGB, *Frontal*-depth, *Top*-RGB, *Top*-depth, HD camera, and audio.

2.3.3.1 Technical Information

The dataset contains 4 synchronized streams of data: 2 RGB-D video feeds, from frontal and top point of views, acquired with *Kinect v1* sensors), 1 RGB video feed from a 1280 $\times$ 720 HD camera, and 1 audio feed captured with a studio microphone. Table 2.4 shows the specific data formats available and Fig. 2.15 shows the cameras placement in the Baxter robot used for the acquisition. The *Frontal* RGB-D camera is mounted on the robot chest to give a frontal view of the user and the ta-

Users	10	
Interaction Type	3	*Point, Show, Speak*
Interactions per User	30	10 of each type. 1 object per interaction.
Object Pool	22	*Apple, Banana, Big Mug, Bowl, Cereal Box, Coke, Diet Coke, Glass, Fork, Ketchup, Kleenex, Knife, Lemon, Lime, Mug, Noodles, Orange, Plate, Pringles, Spoon, Tea Box, Water Bottle*

Table 2.3: Summary of the dataset content

ble. The *Top* RGB-D camera is mounted at the highest point of the robot and has a holistic overview of the scene.

Device	Data	Format
RGB-D Cameras (Frontal & Top)	RGB frames	640 x 480 JPEG
	Depth frames	640 x 480 PNG
HD Camera	RGB frames	1280 x 720 JPEG
Microphone	Audio file	44.1kHz Stereo WAV

Table 2.4: Dataset format specifications

2.3.3.2 Annotations

The dataset annotations include the list of the objects each user interacted with, the first uttered word (which is either "this", "that" or "the"), and the label of the object in question for each interac-

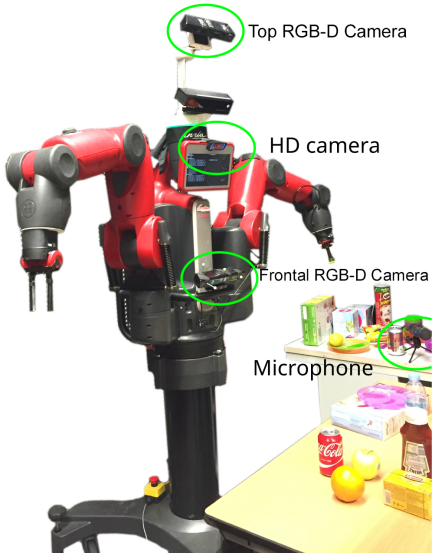


Figure 2.15: Baxter robot used to acquire the dataset. The three cameras and the microphone locations are highlighted.

tion. Additionally, each frame is timestamped (using ROS*) and labeled with the type of interaction (*Point*, *Show*, *Speak*).

2.3.4 Learning from Multimodal Interactions

This section presents the proposed pipeline for object learning by leveraging different data modalities that capture natural interactions. Our pipeline is composed of three modules, summarized in Fig. 2.12:

- **Interaction Recognition.** The extraction of candidate object patches depends on the interaction type (*Point*, *Show*, *Speak*), so an accurate identification of the interaction is crucial.
- **Target Object Detection.** For each interaction type we propose a specific algorithm to select the candidate image patches that are likely to contain the object.
- **Object Model Learning.** The candidate patches from the previous step are used as training examples for supervised learning (the class labels coming from the users' speech).

Our main contributions lie in the first two modules, as once we have extracted the target object patches we can use standard object model learning algorithms. We aim to demonstrate the benefits

*http://ros.org/

of the multiple data sources and the feasibility of learning from natural interaction. The next subsections detail each module.

2.3.4.1 Multimodal Interaction Recognition

Classifying the type of interaction performed by a person using only visual data is considerably challenging. The work of Abdullah and Noah (2008) shows that the combination of language and vision can lead to a substantial improvement. Our interaction recognition module uses visual and language features in a nested SVM-based classification.

Language Features We use a simple language feature consisting of the first word of the user's narration. In our dataset this word is either *this* or *that* for *Point* and *Show* interactions or any other word for the more descriptive *Speak* interaction. This feature is not discriminative enough to separate the three interaction classes, as we show in Fig. 2.16. It clearly separates *Speak* interactions, but cannot differentiate between *Point* and *Show*. Separating *Speak* is particularly valuable, as there are no specific visual patterns associated with this interaction.

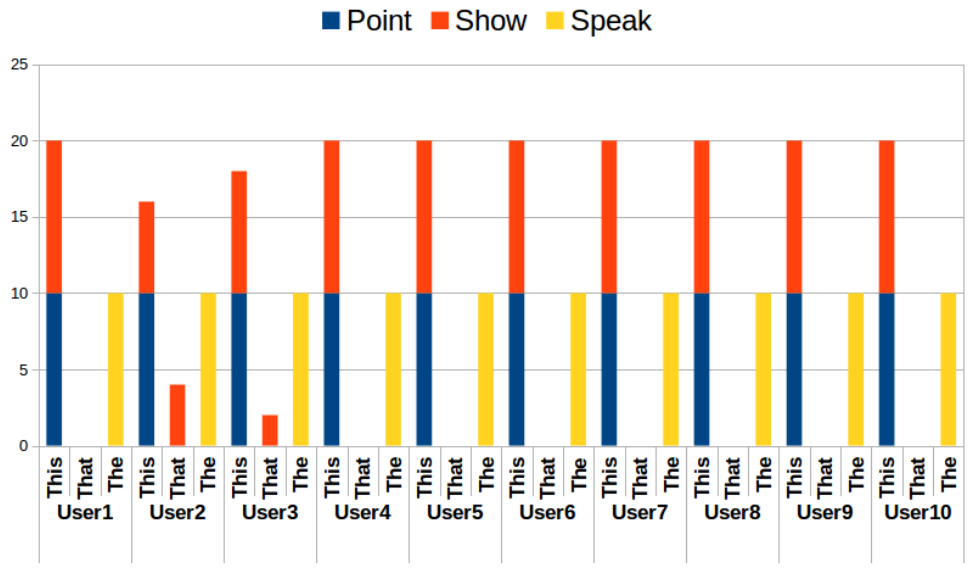


Figure 2.16: Language feature occurrences in all recordings, per type of interaction and per user.

Visual Features Before computing the visual features, in order to focus on the user and table regions, we remove the background using two strategies: a standard background removal procedure, based on sliding-window average of all the previous frames, and a depth map based filter, where we remove all image pixels with a depth value over a threshold of $1.7m$ (based on the distance to the user and the table). We apply these two filters on the image and run a sliding-window filter (window size 100x100 pixels, step of 10) over the masked image to reject windows where more than 30% of the pixels were removed by either one of these filters. Then we compute the visual descriptors on the accepted windows. We evaluate two different descriptors:

- *Color histograms* $HC = [H_r \ H_g \ H_b]$, with $H_i = \sum_{x,y} p_i(x,y) \bmod B$, where p_i is pixel i component value and B the number of bins.
- *Histogram of Gradients* (*HOG*), as described in Dalal and Triggs (2005).

Interaction Recognition We propose the following interaction recognition module, using the aforementioned language and visual features, based on two nested classifiers:

1. Binary discrimination between *Speak* videos and the other two types using the language features.
2. SVM classification into *hand* vs *no-hand* classes of sliding window-based patches, trained with random patches of the dataset and manually selected patches of hands. This step only uses the *HC* descriptor due to its high efficiency and good performance at removing most of the *no-hand* patches.
3. SVM classification of resulting *hand* patches into *Point* or *Show* classes. Here we use both the *HC* and the *HOG* descriptors.
4. Assign a label, *Point* or *Show*, to each video according to the label obtained by the majority of its frames. All windows from each video are labeled as that action for the next step.

2.3.4.2 Target Object Detection

The goal of this module is to extract image patches that are likely to contain the target object we want to model. Based on the results from the previous module, in particular *hand* patches from *Point* or *Show* classes, we propose two algorithms to segment the target object: one using the Frontal

RGB-D camera only (named "Single-Camera") and another using the two RGB-D cameras ("Two-Cameras").

Single-Camera algorithm We start by using SLIC (Achanta et al., 2012) superpixels to segment the image and determine the object area efficiently. We propose different strategies (as detailed in Algorithm 3) to extract the target object patch depending on the interaction type:

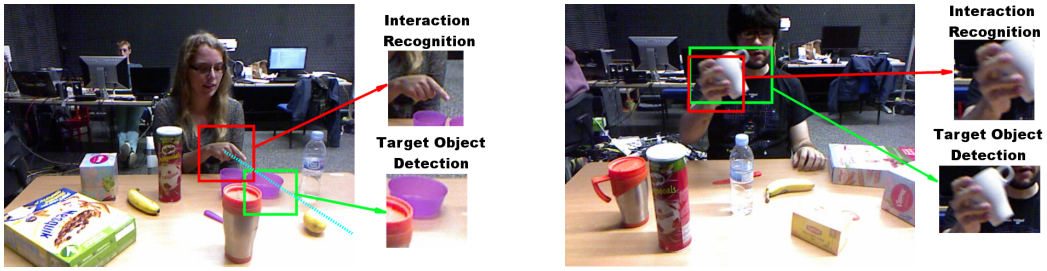
- *Point interaction*: First, we estimate the pointing direction using the first-order moments of the hand superpixels. After that, we select the scene superpixel that intersects with the pointing vector. We extract the image patch that completely contains the intersecting superpixel. See Fig. 2.17(a) for an example.
- *Show interaction*: First, we estimate the hand superpixel orientation as the direction of its first-order moment. We assume that the object is aligned with the hand. Then we extract the image patch that contains the hand superpixel and the neighbouring superpixel following its orientation. See Fig. 2.17(b) for an example.

Algorithm 3 Single-Camera. Target object detection using the Frontal RGB-D camera.

```
Require: Video_RGB-D_Frontal,interaction,Hand_Pos
  if interaction == 'Point' then
    for each Frame in Video_RGB-D_Frontal do
      point_direction = get_pointing_direction(Hand_Pos)
      SuperPixels = SLIC(Frame)
      Sp_intersect = get_intersection(SuperPixels,point_direction)
      Patch = Patch_SP(Frame,Sp_intersect)
      add_patch(Patch)
    Patches = get_patches(Patch) return Patches
  else
    for each Frame_F in Video_RGB-D_Frontal do
      Orientation = get_orientation(Frame,Hand_Pos)
      Patch = expand_Patch(Frame,Hand_Pos,Orientation)
      add_patch(Patch)
    Patches = get_patches() return Patches
```

Two-Cameras algorithm Our proposal leverages the two different points of view of the cameras in the MHRI dataset.

In the cases of *Show* interaction the object is easy to find in the Frontal camera because there is usually no occlusion. Here we are considering the more interesting case of *Point* interactions.



(a) Point Example **(b)** Show Example

Figure 2.17: Target Object Detection using **Single-Camera** algorithm. Two examples from different interaction types. The red box indicates the patch used to recognize the interaction type (hand patch) and the green box indicates the selected target object patch following the corresponding strategy. The dashed line in (a) is the estimated pointing direction.

First, we search for an object candidate in the Top camera in the initial frames before the actual interaction has started. The Top camera allows us to get an accurate object pre-segmentation by subtracting the table plane. In this top view the objects are not affected by occlusions (they are in the frontal view). We have calibrated the table plane homography, so we can map approximately the objects from the Top to the Frontal camera. Once the candidates have been mapped, we calculate the closest object that intersects with the pointing direction in each frame. The object with the most votes within the video is the chosen one, and the image patches containing it are used as training samples. The details can be found in Algorithm 4. Fig 2.18 shows an illustrative example.

2.3.4.3 Object Model Learning

In order to evaluate the proposed target object detection in the context of the presented dataset, we implemented a standard object model learning approach:

- *Descriptors.* We use *Color histograms* (*HC*) and *Bag of Words* (*BoW*), following Nister and Stewenius (2006). The BoW model is built from ORB (Rublee, Rabaud, Konolige, & Bradski, 2011) extracted on images from the Washington dataset (K. Lai et al., 2011). We used a standard k-means to build the vocabulary (with k=1000).
- *Classifiers.* We evaluate two classifiers: Support Vector Machines (**SVM**) and Nearest Neighbors (**NN**).

We disregarded naive Convolutional Neural Nets (CNN) as baseline because (a) the data given in our domain would not suffice to train a model from scratch and (b) transfer learning from a large

Algorithm 4 Two-Cameras. Target Object Detection using the two RGB-D cameras (Frontal and Top).

```
Require: Video_RGB-D_Frontal,Video_RGB-D_Top,interaction,Hand_Pos
  Candidates = Calculate_Candidates(Video_RGB-D_Frontal,Video_RGB-D_Top)
  if interaction == 'Point' then
    for each Frame in Video_RGB-D_Frontal do
      point_direction = get_pointing_direction(Hand_Pos)
      for each Candidate in Candidates do
        dist,Inline = Intersect(Candidate,point_direction)
        if  Inline && get_min_dist()> dist then
          set_candidate(Candidate,dist)
      add_vote(Candidate)
    Winner = get_winner()
    Patches = extract_patches(Winner) return Patches
  else
    for each Frame_F in Video_RGB-D_Frontal do
      Orientation = get_orientation(Frame,Hand_Pos)
      Patch = expand_Patch(Frame,Hand_Pos,Orientation)
      add_patch(Patch)
    Patches = get_patches() return Patches
  function Calculate_Candidates(RGB-D_Frontal, RGB-D_Top)
    Homography = get_Homography()
    for  First five Framesof RGB-D_Frontal, RGB-D_Top do
      Plane,ϑ = Calculate_plane(Frame_Top)
      Frame_Rotated = Rotate(Frame_Top,−ϑ)
      Area_Cropped = Extract_Plane_Crop(Frame_Top,Plane)
      Blobs = get_blobs(Area_Cropped)
      Projected_Blob = get_projection(Homography,Blobs)
      SuperPixels = SLIC(Frame_Frontal)
      Candidates = obtain_candidates(SuperPixels,Projected_Blob) return Candidates
```

object recognition dataset could also fail, as we are dealing with significant occlusions and imperfect segmentations.

2.3.5 Experimental results

This section presents several experiments to demonstrate the challenges of the proposed MHRI dataset. The following experiments validate our pipeline to recognize interaction types and automatically segment target object patches. Finally, it is also our aim to establish a baseline for the dataset.

In the following experiments, we consider four types of patches, all of them containing approximately one object:

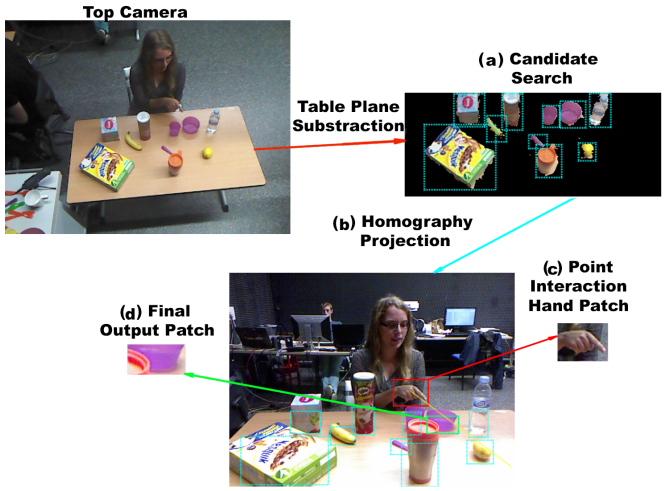


Figure 2.18: Target Object Detection using **Two-Cameras**: (a) extract candidate objects in the Top camera, (b) project them on the Frontal camera, (c) find pointing direction from the hand patch in Frontal camera and (d) select object candidate that intersects with the pointing direction.

- *Washington Patches* which contain correctly labeled objects from the Washington dataset (K. Lai et al., 2011).
- *Manual Patches* which are manually cropped around objects from MHRI dataset images.
- *Automatic Patches* which are automatically obtained using our target object detection algorithm in the MHRI data.
- *Inspected Patches* which are a subset of the *Automatic Patches* that were visually inspected to verify that both patch and label are correct.

Fig. 2.19 shows examples of the different types of patches. As *Inspected patches* is a subset of *Automatic patches* and their visual appearance is similar, examples of the former are excluded from the figure.

2.3.5.1 Interaction Recognition

In order to demonstrate the benefits of multimodal data, we first classify the interaction type by using only visual data and SVM. Table 2.5(a) shows the confusion matrix.

We augment the model with the speech modality using the first word of the user speech (*this/that/the*), as explained in Sec. 2.3.4. Table 2.5(b) shows the confusion matrix obtained by this classifier, which improves the results for all classes, discriminating the *Speak* interaction and improving *Point* and *Show* from 72.85% to 85.71% and 12.88% to 77.97% respectively.

	Point	Show	Speak
Point	**72.85%**	76.01%	55.46%
Show	12.36%	**12.88%**	20.00%
Speak	14.78%	11.11%	**24.54%**

(a) Vision-Only Classification

	Point	Show	Speak
Point	**85.71%**	22.03%	0.00
Show	14.29%	**77.97%**	0.00
Speak	0.00%	0.00%	**100.00%**

(b) Multimodal Classification

Table 2.5: Interaction recognition accuracy.

(a) *Washington* patches **(b)** *Manual MHRI patches* **(c)** *Automatic MHRI patches*

Figure 2.19: Examples of three types of object patches used in our experiments. Notice the increasing levels of clutter and segmentation noise.

2.3.5.2 Target Object Detection

Our aim in this section is to evaluate the quality of the *Automatic patches* obtained by our algorithm. Fig. 2.19 illustrates, qualitatively, the different visual appearance of the *Washington patches* and *Manual patches*. Notice how the *Automatic patches*, obtained from natural interactions, present several challenges for standard object learning methods. There is clutter around the target object and in most of the *Show* examples the hand significantly occludes the object. In many patches from *Point* interactions the target object is not centered and only partially visible. Table 2.6 shows the average number of patches extracted with our approach for *Point* and *Show* clips.

As previously mentioned, we have evaluated deep learning features to model the target object patches. Unfortunately, the domain where object recognition models are usually trained, such as *ImageNet*, contains mostly clean images of complete objects. As Fig. 2.19c shows, the patches we extract contain mostly partial views of the objects, due to noisy segmentations and occlusions. Us-

User:	1	2	3	4	5	6	7	8	9	10	ALL
Point videos											
#P	68	62	75	31	42	32	55	38	65	60	53 (25)
#F	130	91	99	64	62	72	77	78	81	72	82 (24)
Show videos											
#P	133	110	125	97	124	100	136	149	133	123	123 (29)
#F	183	135	136	114	136	112	148	170	141	132	141(28)

#P: total number of object patches extracted

#F: total number of frames per video for this user

#ALL: average and standard deviation for all users and videos

Table 2.6: Target object patches in videos from each user.

ing existing CNNs to extract features in our patches did not improve our results. Fig. 2.20 shows a few classification examples of MHRI patches which contain object classes included in *AlexNet* model (Krizhevsky, Sutskever, & Hinton, 2012a). We can see that the CNN model fails to recognize the object if the patch does not contain the entire object.

banana rapeseed water-bottle milk-can bowl petri-dish

Figure 2.20: Object labels for different sample patches using a pre-trained CNN (*AlexNet*). It often fails when patches contain only partial views of objects (banana and water bottle) or unexpected points of view (bowl).

2.3.5.3 Object Model Learning

The following experiments are designed to evaluate the performance that learning algorithms can obtain in the MHRI dataset. As already mentioned, the challenges are many. The target object patches extracted automatically are noisy, have occlusions and are of low resolution in comparison to other datasets (see Fig. 2.19).

For the experiments in this section, we used the different types of image patches defined before (*Washington*, *Manual*, *Automatic* and *Inspected*) to train an object classifier. We do 10-fold cross-validation for the MHRI patches, each fold being all the images from one user. This ensures basic

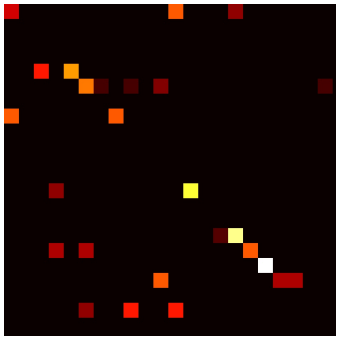


Figure 2.21: Confusion Matrix for the best user in the experiment with *Manual patches* Lighter color is higher accuracy. Black rows correspond to unused objects.

generalization over position, clutter, lighting and user bias in training and testing. The accuracy is averaged over the ten folds for the 22 objects in the MHRI dataset. Although each user only handles a subset of the objects, all the confusion matrices are of the same size, each of the 22 objects per row, following the same object order as Table 2.3. The rows corresponding to unused objects are set to zero (displayed in black).

Washington and Manual patches Both sets of patches serve as baselines to evaluate the full object model pipeline. The *Washington patches* illustrate the domain change of our dataset. And we use the *Manual patches* to set an upper bound for the performance of the *Automatic patches*.

In a first experiment, we train a standard object recognition algorithm (BoW+SVM) using the *Washington patches* and evaluate its performance on *Manual patches* of the MHRI dataset. The average accuracy for the 12 classes that both datasets have in common is close to random (**8.4%**). This demonstrates that even when they share several objects the respective biases of the datasets cause naive approaches with pre-trained models to fail.

	SVM		NN	
	BOW	HC	BOW	HC
Manual	16.5-5.7-30.0	38.5-7.7-55.7(*)	8-5.5-17.8	28.6-8.8-42.2

Accuracy-Std.Deviation-Best Experiment

* Confusion Matrix is shown in Fig. 2.21

Table 2.7: Average accuracy (*Manual patches*).

Our second experiment is to train several classifiers with *Manual patches*, and evaluate their performance in a test set from the same dataset. Table 2.7 shows the results. Notice that their best accuracy, **38.5%**, is considerably higher than the previous one trained on the *Washington patches*, confirming the dataset bias. Notice that SVM shows a better performance than NN, which is why we pick this classifier for the rest of the experiments.

Finally, observe that an accuracy of 38.5% is low for supervised visual classification with manually annotated data. This result shows the challenging nature of our dataset and motivates its release.

Automatic patches We present results for our full pipeline extracting image patches automatically using the approach in section 2.3.4.

Single-Camera vs Two-Cameras: In this experiment we show the benefit of using two cameras, as explained in section 2.3.4.2. Table 2.8 shows the accuracy for our pipeline using one and two cameras. Notice that using two cameras improves the performance, both in average and best experiment accuracy. For the rest of the results in the paper we use the *Two-Cameras* algorithm.

	Frontal Camera		Two Cameras	
	BOW	HC	BOW	HC
Automatic	5.6-2.4-10.2	7.1-4.1-16.0	7.7-6.5-30.5	9.1-6.9-28.0(*)

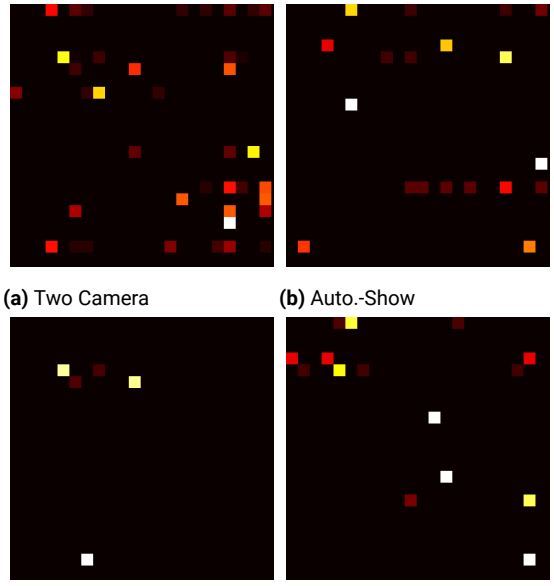
Accuracy-Std.Deviation-Best Experiment

* Confusion Matrix is shown in Fig. 2.22a

Table 2.8: Accuracy (*Single-Camera vs Two-Cameras*, train with *Manual patches*, test with *Automatic patches*).

Inspected patches The aim of the following experiments is to analyze the performance per-interaction type. We use the *Manual patches* for training and the *Automatic patches* and *Inspected patches* for test. Table 2.9 shows the accuracy results for this experiment. *Show* presents a higher accuracy (**9.6%**) than *Point* (**4.5%**) for *Automatic patches*. The patch extraction is more noisy for *Point*, due to the uncertainty associated with the pointing direction estimation. Notice however, that the performance of *Point* is much higher (**20.1%**) if we use *Inspected patches*. The reason is, if the pointing direction is accurately estimated, the *Point* patches are less affected by occlusion and hence the model learned is better. Observe also that, in general, the HC descriptor is better than the BoW one due to the low resolution of the patches and the occlusions.

Finally, Table 2.10 shows the results using *Inspected patches* per user. We would like to highlight the high variability between users. This variability is promising; as the average accuracy is reasonably



(a) Two Camera **(b)** Auto.-Show

(c) Insp.-Point **(d)** Insp.-HC

Figure 2.22: Confusion Matrix for best experiment in the evaluations with *Automatic patches* and *Inspected patches*.

high for certain users, we believe the future research should focus on the challenging cases.

2.3.6 Conclusions

In this paper we have presented an annotated multimodal dataset for object model learning from human-robot interaction. The most remarkable features of the dataset are, first, the synchronized recording of multimodal data (two *RGB-D* cameras, one high resolution *RGB* camera and audio data), and second, the recording of natural interactions for humans (*Point*, *Show* and *Speak*) when teaching new object models. The dataset has a robocentric perspective.

As a second contribution, we have presented a first approach to object learning using multimodal data from natural HRI. Such approach is the initial baseline for the dataset, showing the feasibility and challenges of object learning from natural interactions. Our proposed algorithm also serves to demonstrate how the interaction classification benefits from the use of multimodal data (language and vision). The interaction recognition is a critical step, as the training data has to be extracted differently depending on the particular interaction. We have proposed a target object detection method to extract patches containing the objects, evaluated its performance, and shown its challenges. Finally, we have evaluated the full pipeline against manually extracted data. Our main conclusions are

	Show (75%)		Point (25%)	
	BOW	HC	BOW	HC
Automatic	8.0-8.2-31.1	9.6-6.5-18.4(*)	4.5-7.3-23.0	3.0-6.9-21.3
Inspected	7.6-10.4-35.1	8.6-6.8-19.2	8.4-8.0-27.3	20.1-21.7-66.6(**)

Accuracy-Std.Deviation-Best Experiment

* Confusion Matrix is shown in Fig. 2.22b

** Confusion Matrix is shown in Fig. 2.22c

Table 2.9: *Automatic patches* vs *Inspected patches* object recognition trained with manual patches.

User:	1	2	3	4	5	6	7	8	9	10	ALL
Point videos											
#P	0	11	22	0	53	0	6	36	92	2	22
#H(%)	0	0	0	0	70	0	0	0	59	0	18
Show videos											
#P	47	100	111	47	88	76	114	73	9	116	78
#H(%)	9	43	35	49	10	45	12	19	11	12	24

#P: Average number of object patches accepted after inspection ;

#H: Accuracy classifying them; #ALL: Average for all users and videos

Table 2.10: Accuracy using *Inspected patches*. Confusion Matrix of best example in Fig. 2.22d

the following; First, the domain change is critical, and hence it is impractical to use data from other object datasets in a naive manner. Second, although our approach shows a reasonable performance, there are still considerable challenges in the target object detection and model learning, justifying the relevance of the presented dataset.

In future lines of work, we plan to improve the detection of the direction of the hand, develop new features that take advantages of the depth information, study the use of CNNs for object proposal and create an incremental learning system to discard the noisy or incorrectly labeled patches.

2.4 HoME: a Household Multimodal Environment

We introduce HoME: a **Ho**usehold **M**ultimodal **E**nvironment for artificial agents to learn from vision, audio, semantics, physics, and interaction with objects and other agents, all within a realistic context. HoME integrates over 45,000 diverse 3D house layouts based on the SUNCG dataset, a scale which may facilitate learning, generalization, and transfer. HoME is an open-source, OpenAI Gym-compatible platform extensible to tasks in reinforcement learning, language grounding, sound-based navigation, robotics, multi-agent learning, and more. We hope HoME better enables artificial agents to learn as humans do: in an interactive, multimodal, and richly contextualized setting.

2.4.1 Introduction

Human learning occurs through interaction (K. Fisher, Hirsh-Pasek, & Golinkoff, 2012) and multimodal experience (Landau, Smith, & Jones, 1998; Smith & Yu, 2008). Prior work has argued that machine learning may also benefit from interactive, multimodal learning (Hermann et al., 2017; Oh, Singh, Lee, & Kohli, 2017; de Vries et al., 2017; Gauthier & Mordatch, 2016), termed *virtual embodiment* (Kiela, Bulat, Vero, & Clark, 2016). Driven by breakthroughs in static, unimodal tasks such as image classification (Krizhevsky, Sutskever, & Hinton, 2012b) and language processing (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013), machine learning has moved in this direction. Recent tasks such as visual question answering (Antol et al., 2015), image captioning (Vinyals, Toshev, Bengio, & Erhan, 2017), and audio-video classification (Dhall, Ramana Murthy, Goecke, Joshi, & Gedeon, 2015) make steps towards learning from multiple modalities but lack the dynamic, responsive signal from exploratory learning. Modern, challenging tasks incorporating interaction, such as

---

Section 2.4 was published as

Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, Aaron Courville. **HoME: a Household Multimodal Environment**. In 6th International Conference on Learning Representations (ICLR) 2018, Workshop Track.

To this work we contributed: A major part of the implementation, including a gym-compatible reinforcemenet learning environment for agents; agent control code; part of the physics system; literature research; evaluation of competing environments; performance tests of our code; most diagrams; and a big part of the writing.

Atari (Bellemare, Naddaf, Veness, & Bowling, 2013) and Go (Silver et al., 2016), push agents to learn complex strategies through trial-and-error but miss information-rich connections across vision, language, sounds, and actions. To remedy these shortcomings, subsequent work introduces tasks that are both multimodal and interactive, successfully training virtually embodied agents that, for example, ground language in actions and visual percepts in 3D worlds (Hermann et al., 2017; Oh et al., 2017; Chaplot, Sathyendra, Pasumarthi, Rajagopal, & Salakhutdinov, 2017).

For virtual embodiment to reach its full potential, though, agents should be immersed in a rich, lifelike context as humans are. Agents may then learn to ground concepts not only in various modalities but also in relationships to other concepts, i.e. that forks are often in kitchens, which are near living rooms, which contain sofas, etc. Humans learn by concept-to-concept association, as shown in child learning psychology (Landau et al., 1998; Smith & Yu, 2008), cognitive science (Barsalou, 2008), neuroscience (Nakazawa et al., 2002), and linguistics (Quine, Churchland, & Føllesdal, 2013). Even in machine learning, contextual information has given rise to effective word representations (Mikolov et al., 2013), improvements in recommendation systems (Adomavicius & Tuzhilin, 2011), and increased reward quality in robotics (Jaderberg et al., 2016). Importantly, scale in data has proven key in algorithms learning from context (Mikolov et al., 2013) and in general (Russakovsky et al., 2015; Bojar et al., 2015; Josh Tobin et al., 2017).

To this end, we present HoME: the **Ho**usehold **M**ultimodal **E**nvironment (Figure 2.23). HoME is a large-scale platform* for agents to navigate and interact within over 45,000 hand-designed houses from the SUNCG dataset Song et al., 2017. Specifically, HoME provides:

- 3D visual renderings based on Panda3D;
- 3D acoustic renderings based on EVERT (Laine, Siltanen, Lokki, & Savioja, 2009), using ray-tracing for high fidelity audio;
- semantic image segmentations and language descriptions of objects;
- physics simulation based on Bullet, handling collisions, gravity, agent-object interaction, and more;
- multi-agent support; and
- a Python framework integrated with OpenAI Gym (Brockman et al., 2016).

HoME is a general platform extensible to many specific tasks, from reinforcement learning to language grounding to blind navigation, in a real-world context. HoME is also the first major interactive platform to support high fidelity audio, allowing researchers to better experiment across

*Available at https://home-platform.github.io/

modalities and develop new tasks. While HoME is not the first platform to provide realistic context, we show in following sections that HoME provides a more large-scale and multimodal testbed than existing environments, making it more conducive to virtually embodied learning in many scenarios.

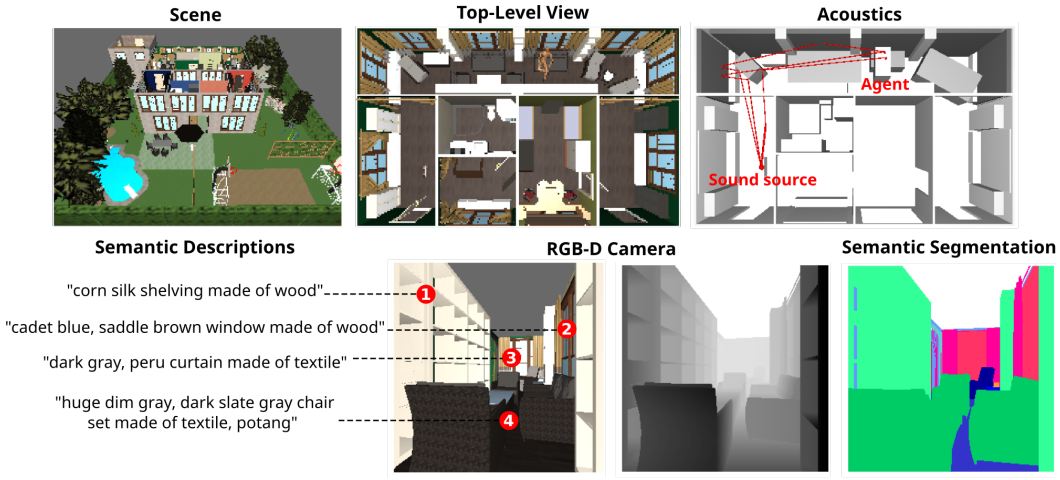


Figure 2.23: A single example that demonstrates HoME's features.

2.4.2 Related work

The AI community has built numerous platforms to drive algorithmic advances: the Arcade Learning Environment (Bellemare et al., 2013), OpenAI Universe ("OpenAI Universe," 2016), Minecraft-based Malmo (Johnson, Hofmann, Hutton, & Bignell, 2016), maze-based DeepMind Lab (Beattie et al., 2016), Doom-based ViZDoom (Kempka, Wydmuch, Runc, Toczek, & Jaśkowski, 2016), AI2-THOR (Y. Zhu et al., 2017), Matterport3D Simulator (P. Anderson et al., 2017) and House3D (Anonymous, 2018). Several of these environments were created to be powerful 3D sandboxes for developing learning algorithms (Johnson et al., 2016; Beattie et al., 2016; Kempka et al., 2016), while HoME additionally aims to provide a unified platform for multimodal learning in a realistic context (Fig. 2.24). Table 2.11 compares these environments to HoME.

The most closely related environments to HoME are House3D, AI2-THOR, and Matterport3D Simulator, three other household environments. House3D is a concurrently developed environment also based on SUNCG, but House3D lacks sound, true physics simulation, and the capacity for agents to interact with objects — key aspects of multimodal, interactive learning. AI2-THOR and Matterport3D Simulator focus specifically on visual navigation, using 32 and 90 photorealistic

houses, respectively. HoME instead aims to provide an extensive number of houses (45,622) and easy integration with multiple modalities and new tasks.

Other 3D house datasets could also be turned into interactive platforms, but these datasets are not as large-scale as SUNCG, which consists of 45622 house layouts. These datasets include Stanford Scenes (1723 layouts) (M. Fisher, Ritchie, Savva, Funkhouser, & Hanrahan, 2012), Matterport3D (A. Chang et al., 2017) (90 layouts), sceneNN (100 layouts) (Hua et al., 2016), SceneNet (57 layouts) (Handa, Pătrăucean, Stent, & Cipolla, 2016), and SceneNet RGB-D (57 layouts) (McCormac, Handa, Leutenegger, & Davison, 2017). We used SUNCG, as scale and diversity in data have proven critical for machine learning algorithms to generalize (Russakovsky et al., 2015; Bojar et al., 2015) and transfer, such as from simulation to real (Josh Tobin et al., 2017). SUNCG's simpler graphics also allow for faster rendering.

Figure 2.24: 3D environments (Left to right): DeepMind Lab, Malmo, VizDoom, HoME. Other environments focus on algorithmic challenges; HoME adds a full context in which to learn concepts.

2.4.3 HoME

Overviewed in Figure 2.23, HoME is an interactive extension of the SUNCG dataset (Song et al., 2017). SUNCG provides over 45,000 house layouts containing over 750,000 rooms and sometimes multiple floors. These houses were hand-designed using Planner5D *, an online interior design interface, and filtered down to valid house configurations via a cleaning task based on the majority vote of three Mechanical Turkers. To use a smaller number of higher quality house configurations in HoME, one may further restrict houses to only those approved unanimously. Within these rooms, of which there are 24 kinds, there are objects from among 84 categories and on average over 14 objects per room. As shown in Figure 2.25, HoME consists of several, distinct components built on SUNCG that can be utilized individually. The platform runs faster than real-time on a single-core CPU, enables GPU acceleration, and allows users to run multiple environment instances in parallel. These features facilitate faster algorithmic development and learning with more data. HoME provides an OpenAI Gym-compatible environment which loads agents into randomly selected houses

*https://planner5d.com/

Environment	3D	Context	Large-scale	Fast	Customizable	Physics	Acoustics	HiRes
Atari (Bellemare, Naddaf, Veness, & Bowling, 2013)				✓				
OpenAI Universe ("OpenAI Universe," 2016)	✓	✓	✓		✓			
Malmo (Johnson, Hofmann, Hutton, & Bignell, 2016)	✓		✓	✓	✓			
DeepMind Lab (Beattie et al., 2016)	✓			✓	✓			
VizDoom (Kempka, Wydmuch, Runc, Toczek, & Jaśkowski, 2016)	✓			✓	✓			
AI2-THOR (Y. Zhu et al., 2017)	✓	✓				✓		✓
Matterport3D Simulator (P. Anderson et al., 2017)	✓	✓		✓				✓
House3D (Anonymous, 2018)	✓	✓	✓	✓	✓			
HoME	✓	✓	✓	✓	✓	✓	✓	

Table 2.11: A comparison of modern environments with HoME. **3D**: Supports 3D settings. **Context**: Provides a realistic context. **Large-scale**: Thousands of readily available environments. **Fast**: Renders quickly. **Customizable**: Adaptable towards various, specific tasks. **Physics**: Supports rigid body dynamics and external forces (gravity, collisions, etc.) on agents and objects. **Acoustics**: Renders audio. **HiRes**: High-quality lifelike visual rendering.

and lets it explore via actions such as moving, looking, and interacting with objects (i.e. pick up, drop, push). HoME also enables multiple agents to be spawned at once. The following sections detail HoME's core components.

2.4.3.1 Rendering engine

The rendering engine is implemented using Panda3D (Goslin & Mine, 2004), an open-source 3D game engine which ships with complete Python bindings. For each SUNCG house, HoME renders RGB+Depth scenes based on house and object textures (wooden, metal, rubber, etc.), multi-source lighting, and shadows. The rendering engine enables tasks such as vision-based navigation, imitation learning, and planning.

This module provides: RGB image (with different shader presets), depth image.

2.4.3.2 Acoustic engine

The acoustic engine is implemented using EVERT*, which handles real-time acoustic ray-tracing based on the house and object 3D geometry. EVERT also supports multiple microphones and sound sources, distance-dependent sound attenuation, frequency-dependent material absorption and reflection (walls muffle sounds, metallic surfaces reflect acoustics, etc.), and air-absorption based on atmospheric conditions (temperature, pressure, humidity, etc.). Sounds may be instantiated artificially or based on the environment (i.e. a TV with static noise or an agent's surface-dependent footsteps).

This module provides: stereo sound frames for agents w.r.t. environmental sound sources.

2.4.3.3 Semantic engine

HoME provides a short text description for each object, as well as the following semantic information:

- **Color**, calculated from object textures and discretized into 16 basic colors, ~130 intermediate colors, and ~950 detailed colors†.
- **Category**, extracted from SUNCG object metadata. HoME provides both generic object categories (i.e. "air conditioner," "mirror," or "window") as well as more detailed categories (i.e. "accordion," "mortar and pestle," or "xbox").
- **Material**, calculated to be the texture, out of 20 possible categories ("wood," "textile," etc.), covering the largest object surface area.
- **Size** ("small," "medium," or "large") calculated by comparing an object's mesh volume to a histogram of other objects of the same category.
- **Location**, based on ground-truth object coordinates from SUNCG.

With these semantics, HoME may be extended to generate language instructions, scene descriptions, or questions, as in (Hermann et al., 2017; Oh et al., 2017; Chaplot et al., 2017). HoME can also provide agents dense, ground-truth, semantically-annotated images based on SUNCG's 187 fine-grained categories (e.g. bathtub, wall, armchair).

This module provides: image segmentations, object semantic attributes and text descriptions.

*https://github.com/sbrodeur/evert

†Colors based on a large-scale survey by Randall Munroe, including relevant shades such as "macaroni and cheese" and "ugly pink," https://blog.xkcd.com/2010/05/03/color-survey-results/

2.4.3.4 Physics engine

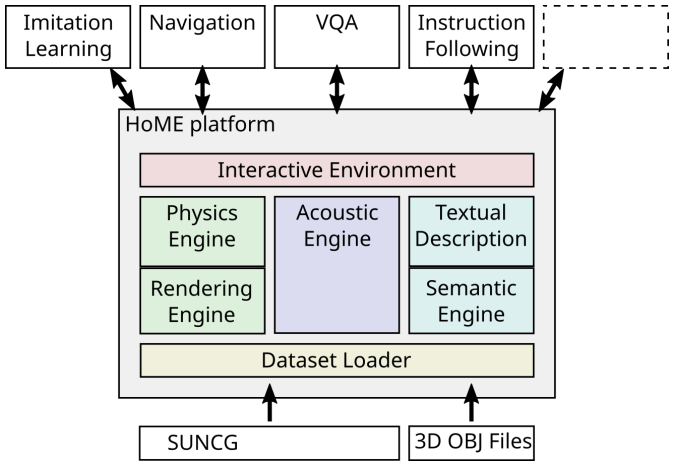


Figure 2.25: HoME's core components which rely on an underlying dataset loader. These components can be used independently and enable HoME to support various specific tasks.

The physics engine is implemented using the Bullet 3 engine*. For objects, HoME provides two rigid body representations: (a) fast minimal bounding box approximation and (b) exact mesh-based body. Objects are subject to external forces such as gravity, based on volume-based weight approximations. The physics engine also allows agents to interact with objects via picking, dropping, pushing, etc. These features are useful for applications in robotics and language grounding, for instance.

This module provides: agent and object positions, velocities, physical interaction, collision.

2.4.4 Applications

Using these engines and/or external data collection, HoME can facilitate tasks such as:

- Instruction Following: An agent is given a description of how to achieve a reward (i.e. "Go to the kitchen." or "Find the red sofa.");
- Visual Question Answering: An agent must answer an environment-based question which might require exploration (i.e. "How many rooms have a wooden table?");
- Dialogue: An agent converses with an oracle with full scene knowledge to solve a difficult task;

*https://github.com/bulletphysics/bullet3

- Pied Piper: One agent must follow another specific agent, out of several, each making specific sounds. HoME's advanced acoustics allow agents with multichannel microphones to perform sound source localization and disentanglement for such a task; and
- Multi-agent communication: Multiple agents communicate to solve a task and maximize a shared reward. For example, one agent might know reward locations to which it must guide other agents.

2.4.5 Conclusion

Our **Ho**usehold **M**ultimodal **E**nvironment (HoME) provides a platform for agents to learn within a world of context: hand-designed houses, high fidelity sound, simulated physics, comprehensive semantic information, and object and multi-agent interaction. In this rich setting, many specific tasks may be designed relevant to robotics, reinforcement learning, language grounding, and audio-based learning. HoME's scale may also facilitate better learning, generalization, and transfer. We hope the research community uses HoME as a stepping stone towards virtually embodied, general-purpose AI.

Suddenly, the familiar view of our surroundings is transformed in a strange, delightful, or alarming way: it appears to us in a new light, takes on a special meaning.

Albert Hofman (from "LSD my Problem Child")

3

Requiem for a Shape

Let's go one level lower. So far, we assumed that we can easily model our surroundings and use that for a simulation. In this chapter we'd like to strip away this ability too. This chapter investigates how we can create a complex 3D world through observing it with a 2D camera. In practice this is often done with LIDAR and other expensive sensors but in the spirit of this work, we're focusing on a solution that works with existing 2D cameras.

Modelling 3D scenes from 2D images is a long-standing problem in computer vision with implica-

---

Chapter 3 was submitted to ICLR 2019 as

Saj Rajeswar, Fahim Mannan, David Vazquez, Florian Golemo, Derek Nowrouzezahrai, Aaron Courville. **Pix2Scene: Learning Implicit 3D Representations from Images**. Submitted to 7h International Conference on Learning Representations (ICLR) 2019.

To this work we contributed: Discussions and ideas to improve the method over the course of the project; the initial architecture and rendering pipeline; a large part of the writing and some of the diagrams and tables in this article; some experiments; a higher fidelity rendering process for mass data generation; the implementation of the quantitative tests; and all the baselines for the quantitative tests.

tions in, e.g., simulation and robotics. We propose **pix2scene**, a deep generative-based approach that implicitly models the geometric properties of a scene from images. Our method learns the depth and orientation of scene points visible in images. Our model can then predict the structure of a scene from various, previously unseen view points. It relies on a bi-directional adversarial learning mechanism to generate scene representations from a latent code, inferring the 3D representation of the underlying scene geometry. We showcase a novel differentiable renderer to train the 3D model in an end-to-end fashion, using only images. We demonstrate the generative ability of our model qualitatively on both a custom dataset and on ShapeNet. Finally, we evaluate the effectiveness of the learned 3D scene representation in supporting a 3D spatial reasoning.

3.1 Introduction

Understanding and modelling the structure and 3D properties of the real world is a fundamental task with broad applications in engineering, simulation, when training artificial agents and any more general scene inference task. Given that the majority natural scene data is available exclusively in the form of 2D images and videos, the ability to build knowledge about the 3D structure underlying these images would be of great utility to scene understanding-based applications.

A powerful strategy employed to recover 3D structure from images is to model object understanding from a large repository of 3D CAD models (A. X. Chang et al., 2015), either by learning the distribution of voxel-based geometric primitives (Z. Wu et al., 2015; J. Wu, Zhang, Xue, Freeman, & Tenenbaum, 2016) or by employing retrieval mechanisms (Choy, Xu, Gwak, Chen, & Savarese, 2016). These strategies can be extended from single objects to entire scenes by further modelling plausible configurations and alignment of the individual objects (again, directly from the image data). On this axis, the state-of-the-art either explicitly models the physical constraints of object-to-scene placement (Huang et al., 2018) or directly infers these relationships (Izadinia, Shan, & Seitz, 2017). Retrieval-based methods typically lead to scene reconstructions that are realistic and detailed, but limited to only the objects available in the source 3D repository; as such, the resulting 3D scene representations may vary significantly from the actual scene depicted in the input images. The generalization of such strategies to novel or complex scene configurations remains an open problem.

Deep generative models, such as variational autoencoders (VAEs) (Diederik P. Kingma & Welling, 2014) and generative adversarial networks (GANs) (Goodfellow et al., 2014), can excel at modeling high-dimensional data, particularly the distribution of natural images (Karras, Aila, Laine, & Lehtinen, 2018; J.-Y. Zhu, Park, Isola, & Efros, 2017; Radford, Metz, & Chintala, 2015). Recent work using these approaches demonstrate the ability to synthesize novel 3D scenes directly from images:

Kulkarni, Whitney, Kohli, and Tenenbaum, 2015 learn an interpretable and disentangled graphical representation of an underlying image distribution, using VAEs for specific transformations such as out of axis rotation; Rezende et al., 2016 propose a deep generative mechanism to infer the actual 3D representation from 2D images, without any form of direct 3D supervision. Despite promising recent progress, learning high-quality multi-scale representations of arbitrary 3D scene data from images remains a challenge.

We propose **pix2scene**, a deep generative-based approach for modelling the 3D structure of a scene directly from images. We rely on predicting colour and depth as a function of viewpoint in order to infer and distinguish between surface orientation (i.e., normals) and surface depth information. Thus, our 3D representation of a scene is *implicit*: as the viewpoint changes, the model re-generates appropriate depth and colour information that remains consistent with an underlying (singular) 3D scene structure. This allows us to extrapolate scene information to novel, unobserved viewpoints.

We base our model on an Adversarially Learned Inference (ALI) approach (Dumoulin et al., 2016). ALI extends GAN frameworks by learning an additional inference mechanism: while ALI's encoder maps training samples to latent codes, its decoder plays the role of a more traditional GAN generator, mapping from the latent space to image space. We extend this adversarial framework with 3D priors on the smoothness of depth maps, constraining the generated 3D outputs. This combination of adversarial learning with 3D priors allows us to learn interpretable representations of the 3D scene geometry, directly from 2D images and without any explicit supervision. Our representation is a novel, view-dependent combination of depth maps and (implicit) surface orientations.

We present several contributions to the field of 3D scene generation. First, unlike prior art, our 3D representation of real-world scenes relies on *view-space surface elements* analogous to the *surfel represnetation* used for rendering in computer graphics (Pfister, Zwicker, van Baar, & Gross, 2000). These *surf* ace *el*ement*s* comprise the position, orientation/normal, and material reflectance properties of surface patches in a 3D scene. Secondly, we apply a novel adversarial learning-based framework to directly generate these surfel representations, along with an implicit 3D representation of the entire scene, and all this only from input 2D images. Our approach relies fundamentally on a novel surfel-based differentiable 3D renderer to synthesize 2D images from the 3D surfel representations output by our generative model. To the best of our knowledge, this is the first architecture able to learn non-trivial 3D scene models, such as ones composed using objects from ShapeNet (A. X. Chang et al., 2015), directly from images. One key novelty of our approach is that we train our model in a completely self-supervised manner, without the need for any depth-based supervision.

3.2 Related Work

The generation and reconstruction of 3D scenes from images has been studied extensively in the computer vision and graphics communities (Chaudhuri, Kalogerakis, Guibas, & Koltun, 2011; A. X. Chang et al., 2015; Rezende et al., 2016; Soltani, Huang, Wu, Kulkarni, & Tenenbaum, 2017). Early (non-parametric) 3D generative models tried to combine object parts in order to form new objects, in a probabilistic manner (Chaudhuri et al., 2011; Kalogerakis, Chaudhuri, Koller, & Koltun, 2012). Current methods more often rely on a top-down approach: i.e., shape prior-based methods (Chaudhuri et al., 2011; Choy et al., 2016) learn mappings from 2D observations of single objects to a suitable 3D shape prior, retrieved from a repository of CAD models. These methods can be extended to larger scenes by inferring the 3D layout of the individual objects a priori (Lee, Badrinarayanan, Malisiewicz, & Rabinovich, 2017) or by considering the layout estimation problem in an end-to-end model (Izadinia et al., 2017; Huang et al., 2018).

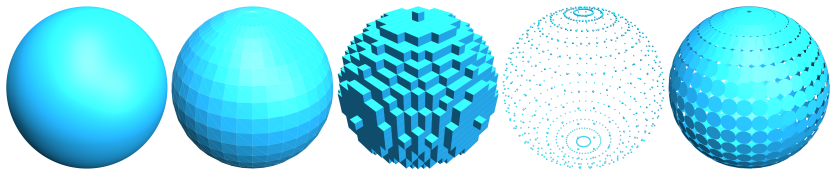
The majority of existing methods that estimate 3D structure are based on volumetric or annotation-based representations of a scene (J. Wu, Wang, et al., 2016; J. Wu, Xue, et al., 2016; Soltani et al., 2017). Volumetric (i.e., voxel-based) representations face scalability challenges and, as such, often result in coarse/low-resolution output. Furthermore, volumetric representations somewhat artificially force the generative model into the difficult (and often non-identifiable) task of inferring the unobserved internal volumetric structure of objects (and, more broadly, entire scene elements).

Our work bears some conceptual similarities to Kulkarni et al., 2015 which casts the 3D reconstruction problem as a more traditional inverse graphics task. By using VAEs, they learn a representation of objects that disentangles factors of variations from images (e.g., object pose and configuration). Unlike this approach, ours is fully unsupervised and implicitly generates 3D scene structure from single images. Our mechanism learns a latent representation for the underlying 3D scene, which we later employ to render the scene from novel views and under novel lighting conditions. Similarly, Rezende et al., 2016 infer 3D configurations, adopting a probabilistic inference framework to build a generative model for 3D. Their work combines standard projection mechanisms with gradient estimation methods. Of particular note, their approach requires multiple runs with mechanisms such as REINFORCE (Williams, 1992) in order to infer gradients from the projection layer. This limits the scalability when considering mesh- and/or voxel-based output representations. Our approach, on the other hand, can adapt to arbitrary scene configurations and does not suffer from any such scalability restrictions .

3.3 Method

3.3.1 3D Representation: Surfels

Standard 3D representations such as voxels, meshes or points clouds present different challenges for generative models (Kobbelt & Botsch, 2004). Voxels make the problem tractable but do not scale well with the scene's resolution. The graphical nature of meshes makes it difficult to optimize. As for point clouds, the spatial structure is difficult to use effectively and thus it's also difficult to capture the spatial/rotational invariances (Qi, Su, Mo, & Guibas., 2017). Accordingly, we propose to model 3D structure with an implicit surface similar to the surfels (Pfister et al., 2000) representation. This representation is very compact: given a renderer's point of view, we can represent only the part of the 3D surface needed by the renderer. Figure 3.1 illustrates these different representations. For descriptive purpose, surfels are shown as oriented disks, but in general they do not have any shape.



(a) Sphere **(b)** Mesh **(c)** Voxels **(d)** Point cloud **(e)** Surfels

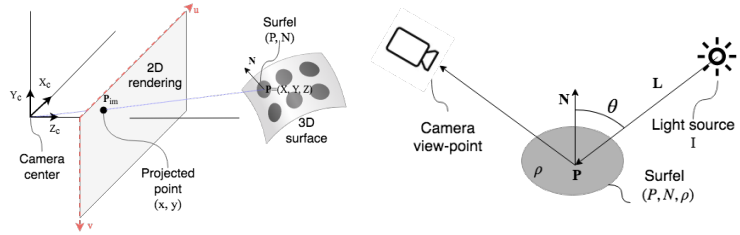
Figure 3.1: 3D representations. Visualization of different computer graphics 3D representations

Surfels are represented as a tuple (P, N, ϱ), where $P = (p_x, p_y, p_z)$ is its 3D position, $N = (n_x, n_y, n_z)$ is the surface normal vector, and $\varrho = (k_r, k_g, k_b)$ is the reflectance of the surface material. Since we are interested in modelling structural properties of the scenes i.e., geometry and depth, we assume that objects in the scene have the uniform material. We represent the surfels in the camera coordinate system. This significantly reduces the number of surfels by considering only the ones that will get projected onto a pixel in the rendered image. Moreover, this allows to reduce the position parameters to only p_z being this the distance along a ray going through the surfel to the center of its pixel.

3.3.2 Differentiable 3D Renderer

As our model's critic operates only on image space, we need to project the generated 3D representations back to the 2D space. In a gradient based optimization setting we therefore need a renderer where each stage of the rendering pipeline is differentiable. By using a differentiable renderer as a

layer in the neural network, we can backpropagate through the entire network and receive a gradient with respect to the surfels. The rendering process can be partitioned into two stages. The first stage finds the mapping between the surfels and the pixels. In the second stage, the renderer computes the color of each pixel. In terms of automatic differentiation, during backpropagation, the first stage directs the gradients only to the surfels that get projected onto the image, and the second stage is differentiable as long as the shading operations are differentiable.



(a) Projection model **(b)** Shading model

Figure 3.2: Differentiable 3D renderer. (a) Surfels are defined by its position P and normal N and projected into its corresponding image point P_{im}. (b) The surfel's color depends on the angle ϑ between the lights I, the surfel's normal N and the reflectance ϱ.

The first stage of the rendering involves finding the mapping between the surfels and the pixels. This requires performing the expensive operation of ray object intersection (See figure3.2a). Therefore our model requires a fast rendering engine as we have to use it in every learning iteration. Conventional ray tracing algorithms are optimized for generating multiple views from the same scene. However in our setting during learning we render only one image from each scene. Moreover ray tracing algorithms require from representing the full scene which is very inefficient as we only represent the part visible by the camera. To solve these issues, our generator proposes one surfel for each pixel in the camera's coordinate system. Our PyTorch implementation of the differentiable renderer can render a 128×128 surfel-based scene in under 1.4 ms on a mobile NVIDIA GTX 1060 GPU.

The color of a surfel depends on the material's reflectance, its position and orientation, and the ambient and point light source colors. See figure3.2b. Given a surface point P_i, the color of its corresponding pixel I_{rc} is given by the shading equation:

$$I_{rc} = \varrho_i (L_a + \sum_j \frac{\mathrm{I}}{k_l \|d_{ij}\| + k_q \|d_{ij}\|^2} L_j \max\left(0, N_i^T d_{ij} / \|d_{ij}\|\right)), \tag{3.1}$$

where ϱ_i is the surface reflectance, L_a is the ambient light's color, L_j is the j^{th} positional light source's color, with $d_{ij} = L_j^{\text{pos}} - P_i$, or the direction vector from the scene point to the point light source, and k_l, k_q being the linear and quadratic attenuation terms respectively.

3.3.3 Pix2Scene Model

The adversarial training paradigm allows the generator network to capture the underlying target distribution by competing with an adversarial critic network, formulated as a min-max game. Pix2scene employs bi-directional adversarial training to model the distribution of surfels from just 2D images.

3.3.3.1 Bi-Directional Adversarial Training

ALI (Dumoulin et al., 2016) or Bi-GAN (Donahue, Krähenbühl, & Darrell, 2016) extends the GAN framework by providing a mechanism for learning a representation of the data distribution in an unsupervised manner. Specifically, in addition to the decoder network G_x, ALI provides an encoder G_z which maps data points $\boldsymbol{x}$ to latent representations $\boldsymbol{z}$. In these bi-directional models, critic D discriminates in both the data space ($\boldsymbol{x}$ versus $G_x(\boldsymbol{z})$), and latent space ($\boldsymbol{z}$ versus $G_z(\boldsymbol{x})$), maximizing the adversarial value function over two joint distributions. The final min-max objective can be written as:

$$\min_G \max_D \mathcal{L}(G, D) := \mathbb{E}_{q(\boldsymbol{x})}[\log(D(\boldsymbol{x}, G_z(\boldsymbol{x})))] + \mathbb{E}_{p(\boldsymbol{z})}[\log(1 - D(G_x(\boldsymbol{z}), \boldsymbol{z}))], \tag{3.2}$$

where $q(\boldsymbol{x})$ and $p(\boldsymbol{z})$ denote encoder and decoder marginal distributions.

3.3.3.2 Modelling depth and Constrained Normal Estimation

Based on the ALI formulation, as depicted in figure 3.3, our model has an encoder network G_x which captures the distribution over the latent space given an image data point $\boldsymbol{x}$. But it also has a decoder network G_z which maps from a fixed latent distribution $p(\boldsymbol{z})$, a standard normal distribution in our case, to the 3D surfel representation. The resulting surfels are then rendered into a 2D image using our differentiable renderer. This image is then given as input to the critic to distinguish from the real image data. Note that the input to the critic comes from the joint space of data with its corresponding latent code, as in ALI.

A straightforward way to model the decoder network could be to learn a conditional distribution to produce the surfel's depth (p_z) and normal (N). But this could lead to inconsistencies between the local shape and the surface normal. For instance, the decoder can fake an RGB image of a 3D shape simply by changing the normals while keeping the depth fixed. Furthermore, real-world surfaces can be considered to be planar in a small neighborhood. Since we only consider surfels that are visible to the camera, the surface normal is constrainted to be in the direction of the camera. Considering the camera to be looking along the $-z$ direction, the estimated normal has the constraint $n_z > 0$.

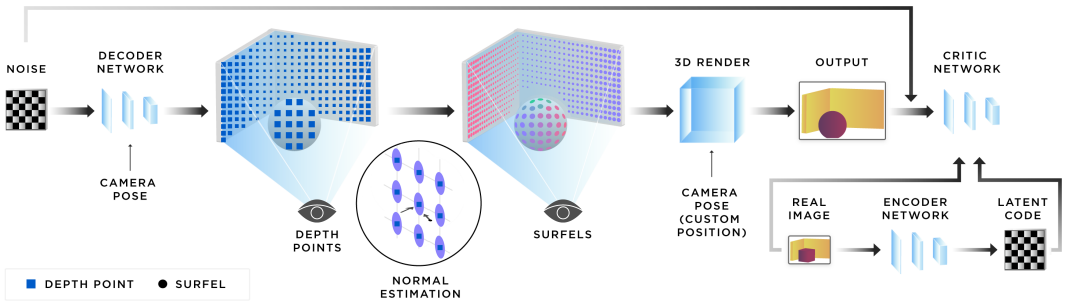


Figure 3.3: Pix2scene model. Pix2scene generates realistic 3D views of scenes by training on 2D images alone. Its decoder generates the surfels depth p_z from a noise vector z conditioned on the camera pose. The surfels normal is estimated from its predicted depth. The surfels are then rendered into a 2D image and together with image samples from the target distribution are fed to the critic.

Therefore, the local surface normal is estimated by solving the following problem for every surfel,

$$\begin{aligned} \|N^T \nabla P\| &= 0 \\ \text{subject to, } \|N\| &= 1 \text{ and } n_z > 0, \end{aligned} \tag{3.3}$$

where the spatial gradient ∇P is computed using the 8-neighbour points, and P is the position of the surfels in the camera coordinate system obtained by backprojecting the generated depth along rays.

This approach makes faking 3D a difficult job for the model by just changing the normals. If the depth is incorrect, the normal-estimator outputs a random set of normals corresponding to the incorrect depth, and these random set of normals would result in an incorrect RGB image, which would in-turn get penalized by the critic. The decoder and the encoder networks would therefore be smoothly updated to obtain realistic depths in order to fool the critic network.

3.3.3.3 Model Training

The Wasserstein-GAN (Arjovsky, Chintala, & Bottou, 2017) formalism provides stable training dynamics using the 1-Wasserstien distance between the distributions. In a high dimensional setting the standard GAN KL divergence could turn out to be too strong a comparison metric. We adopt the gradient penalty setup as proposed in Gulrajani, Ahmed, Arjovsky, Dumoulin, and Courville, 2017 for more robust training, however, we modify the formulation to take into account the bidirectional training.

Architectures of our networks and training hyper parameters are explained in detail in section 3.3.3.4.

We deal with 128 × 128 × 3 resolution for all our experiments. We found Conditional Normaliation (Dumoulin et al., 2016; Perez, Strub, De Vries, Dumoulin, & Courville, 2017) was a better alternate for conditioning the view point than concatenation, which is more traditionally employed for conditional GANs (Mirza & Osindero, 2014). In our training, the affine parameters of the Batch-Normaliation layers (Ioffe & Szegedy, 2015) are replaced by learned representations based on the view point. In order to make sure the reconstructions from the model do not deviate too much from its inputs we also minimize the reconstruction error. Li et al., 2017 argue that minimizing the reconstruction error fixes the identifiability issues with adversarial bi-directional models by minimizing the conditional entropy.

3.3.3.4 Architecture

Pix2scene network architecture is composed by an encoder (See table 3.1), a decoder (See table 3.2), and a critic (See table 3.3). Specifically, the decoder architecture is similar to the generator in DCGAN (Radford et al., 2015) but with LeakyReLU (Mikolov, Deoras, Kombrink, Burget, & Cernockỳ, 2011) as activation function and batch-normalization (Ioffe & Szegedy, 2015). Also, we adjusted its depth and width to accommodate the high resolution images accordingly. In order to condition the camera position on the z variable, we use conditional normalization in the alternate layers of the decoder. We train our model for 60K iterations with a batchsize of 6 with images of resolution 128 × 128 × 3.

3.4 Experiments

Experiments were carefully designed to exhibit the model's understanding of the underlying 3D structure of the 2D input data without any form of annotations.

3.4.1 Implicit 3D Scene Reconstruction

To assess the ability of the network to learn basic 3D shapes from 2D images we trained our model on an extensive set of simulated scenes. The training data consists of rooms with different objects (e.g., teapot, torus, box, sphere, cone) placed in random orientations and positions. Each training image is rendered from a random view sampled uniformly on the positive octant of a sphere containing the room. Technically, the probability of seeing the same configuration of a scene from two different views is near zero. Figure 3.4 shows the input data and its corresponding reconstructions, along with its recovered depth and normal maps. The depth map is encoded in such a way that the

Encoder					
Layer	**Outputs**	**Kernel size**	**Stride**	**BatchNorm**	**Activation**
Input $[x, c]$	$128 \times 128 \times 3$				
Convolution	$64 \times 64 \times 85$	4×4	2	Yes	LeakyReLU
Convolution	$32 \times 32 \times 170$	4×4	2	Yes	LeakyReLU
Convolution	$16 \times 16 \times 340$	4×4	2	Yes	LeakyReLU
Convolution	$8 \times 8 \times 680$	4×4	2	Yes	LeakyReLU
Convolution	$4 \times 4 \times 1360$	4×4	2	No	LeakyReLU
Convolution	$1 \times 1 \times 1$	4×4	1	No	

Table 3.1: Pix2scene encoder architecture

darkest points are closer to the camera. The normal map colors correspond to the cardinal directions (red/green/blue for x/y/z axis respectively). Figure 3.5 showcases similar results on a dataset comprising of rotating boxes in a room. Section 3.4.2 provides quantitative evaluation of reconstructions from the model using surfel based Hausdorff distance measurement.

3.4.2 Evaluation of 3D reconstructions

For evaluating 3D reconstructions, we use the Hausdorff distance (Taha & Hanbury, 2015) as a measure of similarity between two shapes as in Niu, Li, and Xu, 2018. Given two point sets, A and B, the Hausdorff distance is, $\max\left\{\max D_H^+(A, B), \max D_H^+(B, A)\right\}$, where D_H^+ is an asymmetric Hausdorff distance between two point sets. E.g., $\max D_H^+(A, B) = \max D(a, B)$, for all $a \in A$, or the largest Euclidean distance $D(\cdot)$, from a set of points in A to B, and a similar definition for the reverse case $\max D_H^+(B, A)$.

We used Hausdorff distance to compare the surfels positions. Specifically, we measured our model's 3D reconstruction's correspondence with the inputs of a specific camera position on a held-out test data. Table 3.4 shows a quantitative evaluation of the forward and reverse Hausdorff distances on three different datasets. The table also depicts mean squared error of the generated depth map with respect to the input depth map.

Decoder					
Layer	**Outputs**	**Kernel size**	**Stride**	**BatchNorm**	**Activation**
Input $[x, c]$	131×1				
Convolution	$4 \times 4 \times 1344$	4×4	1	Yes	LeakyReLU
Convolution	$8 \times 8 \times 627$	4×4	2	Yes	LeakyReLU
Convolution	$16 \times 16 \times 336$	4×4	2	Yes	LeakyReLU
Convolution	$32 \times 32 \times 168$	4×4	2	Yes	LeakyReLU
Convolution	$64 \times 64 \times 84$	4×4	2	Yes	LeakyReLU
Convolution	$128 \times 128 \times nCh$	4×4	2	Yes	

Table 3.2: Pix2scene decoder architecture.

3.4.3 Implicit 3D Scene Generation

The ShapeNet dataset (A. X. Chang et al., 2015) contains 220,000 3D models classified into 3,135 categories. We used ShapeNetCore which is a subset of ShapeNet that contains 57,386 models from 55 different object categories. To asses the ability of the network to generate new 3D shapes we trained it on 2D images of various objects from 6 classes of the ShapeNet dataset: bowls, bottles, mugs, cans, caps and bags. We have created simple scenes by adding those objects inside a room such that the objects translate randomly relative to the background in each training instance. As shown in figure 3.6, our model is able to generate correct 3D interpretations of the world that are also visually appealing and original. We also trained our model conditionally by giving the class label of the ShapeNet object to the decoder and critic. During inference we conditioned the generator on the object class we wanted to sample from. Figure 3.7 shows the results of conditioning the generator on different classes.

3.4.4 Representation Analysis

In this section we show the richness of pix2scene learned representation. Its learned representations capture the true underlying scene configuration by teasing apart view-based information from scene geometry.

In order to explore the manifold of the learned representations we select two images $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$

Critic					
Layer	**Outputs**	**Kernel size**	**Stride**	**BatchNorm**	**Activation**
Input $[x, c]$	$128 \times 128 \times 6$				
Convolution	$64 \times 64 \times 85$	4×4	2	No	LeakyReLU
Convolution	$32 \times 32 \times 170$	4×4	2	No	LeakyReLU
Convolution	$16 \times 16 \times 340$	4×4	2	No	LeakyReLU
Convolution	$8 \times 8 \times 680$	4×4	2	No	LeakyReLU
Convolution + [z]	$4 \times 4 \times 1360$	4×4	2	No	LeakyReLU
Convolution	$1 \times 1 \times 1$	4×4	1	No	

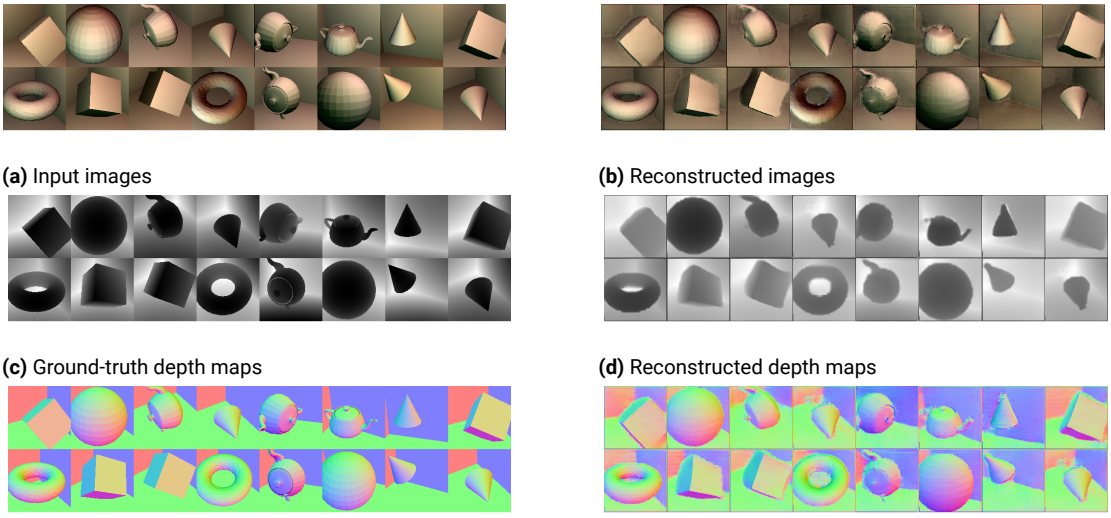
Table 3.3: Pix2scene critic architecture. Conditional version takes image, latent code z and camera position c.

from the held out data, then we linearly interpolate between its encodings $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$ and decode those intermediary points into its corresponding images. Figure 3.8 shows this for two different setups. In each case we can see that our representations capture the geometrical aspects of the scene.

In order to showcase the ability of pix2scene to generate views from unobserved angles of a scene during inference time, we infer the latent code $\boldsymbol{z}$ of a chosen image $\boldsymbol{x}$ and then we decode and render different views while rotating the camera around the scene. Figure 3.9 shows how pix2scene correctly infers the extents of the scene not in view in a consistent manner, demonstrating true 3D understanding of the scene. We show this phenomenon again on different training setups from primitives

Metric	**box in a room (rand rotation)**	**objects in a room (rand rotation)**	**box in a room (rand translation)**
Hausdorff-F	0.102	0.125	0.087
Hausdorff-R	0.183	0.191	0.093
MSE depth map	0.022	0.038	0.032

Table 3.4: Scene reconstruction results. Hausdorff metric on 3D surfel positions and MSE on the depth maps.



(a) Input images

(b) Reconstructed images

(c) Ground-truth depth maps

(d) Reconstructed depth maps

(e) Ground-truth normal maps

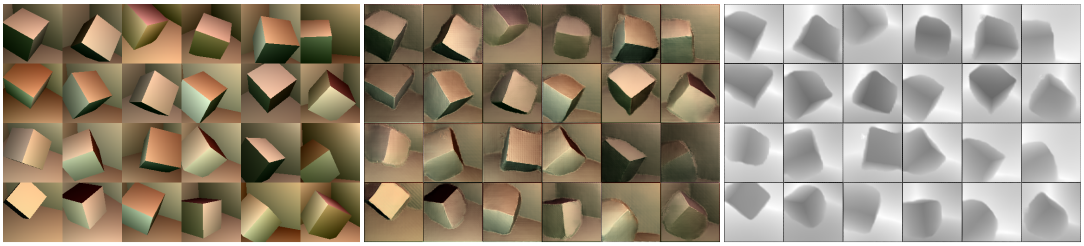
(f) Reconstructed normal maps

Figure 3.4: Scene reconstruction. Left: Input images of rotated objects into a room with its depth and normal groundtruth maps. **Right:** pix2scene reconstructions with its depth and normal maps.

to complex scene configurations

3.4.5 3D-IQ Test Task

We have designed a quantitative evaluation for 3D reconstruction which we refer to as the 3D-IQ test task (3D-IQTT). Some IQ tests, like the Woodcock-Johnson test (Woodcock, Mather, & McGrew, 2001), employ a mental rotation task where the test taker is presented with a Tetris-like shape as a reference and has to find the rotated version of that shape among a few distractors. Analogously, we generated a test where each IQ question instance consists of a reference image, as well as 3 other images, one of which is a randomly rotated version of the reference (see figure 3.12 for an example). The training set is formed by 200k questions where only a few are labelled with the information about the correct answer (i.e., either 5% (10k) or 0.5% (1k) of the total training data). The validation and test sets each contain 100K labelled questions. To verify that our approach is able to learn accurate embeddings of these shapes we first assessed the reconstruction of these shapes qualitatively as shown in figure 3.11.



(a) Input images **(b)** Reconstructed images **(c)** Reconstructed depth maps

Figure 3.5: Scene reconstruction. (a) Input images of rotated cubes into a room. (b) pix2scene reconstructions with its (c) associated depth maps.

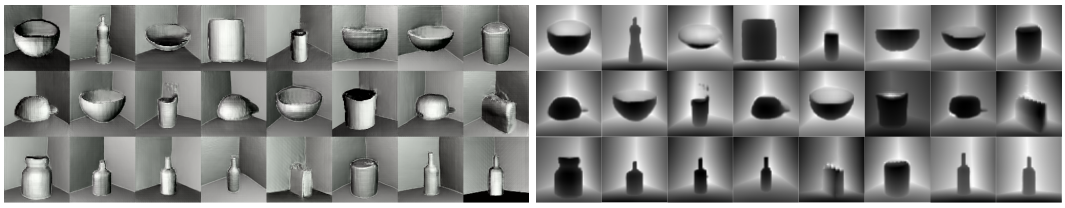


Figure 3.6: Unconditional scene generation. Generated samples from pix2scene model trained on rooms with ShapeNet objects (i.e., bowls, bottles, mugs, cans, caps and bags) located at random positions. **Left:** shaded images; **Right:** depth maps

3.4.6 Semi-supervised classification on the 3D-IQ Test Task

The goal of the 3D-IQTT is to quantify how well we can leverage our generative model to perform 3D spatial reasoning on the test set by using the unlabeled training data to extract the necessary 3D information. The evaluation metric is the percentage of correctly answered *questions*.

For training pix2scene in a semi-supervised setting, in addition to the unlabeled data, we also used the labelled data. The training with the unlabeled samples differs from the approach described for previous experiments as we do not assume that we have the camera position available. Thus, part of the latent vector $\boldsymbol{z}$ encodes the actual 3D object (denoted as $\boldsymbol{z}_{scene}$) and the remainder encodes the rotation (denoted as $\boldsymbol{z}_{view}$). For the supervised training three additional loss terms were added: (a) a loss that enforces the object component to be the same for both the reference object and the correct answer, (b) a loss that maximizes the distance between geometric component of reference and the distractors, and (c) a loss that minimizes the mutual information between $\boldsymbol{z}_{scene}$ and $\boldsymbol{z}_{view}$. Losses (a) and (b) are contained in equation 3.4 where d_i denotes the distractors, $\boldsymbol{x}_{ref}$ is the reference and $\boldsymbol{x}_{ans}$ the correct answer. We interleaved the unsupervised training every 100 steps with complete

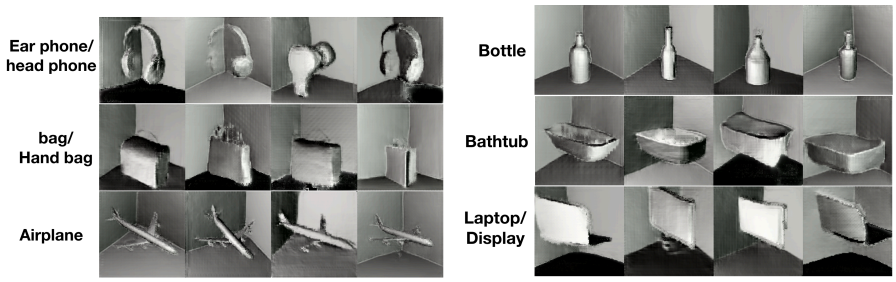


Figure 3.7: Conditional scene generation. Class conditioned generated samples for ShapeNet dataset.

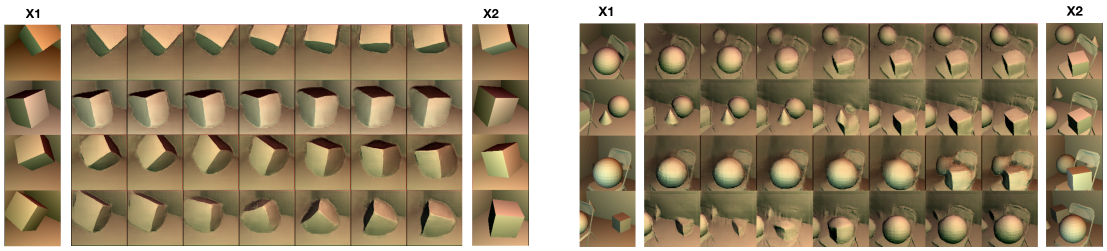


Figure 3.8: Representation analysis. Exploration of the learned manifold of 3D representations. Generated interpolations (middle columns) between two images $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$ (first and last columns).

iterations over the supervised data.

$$\begin{aligned} \mathcal{L} \quad (\boldsymbol{x}_{ref}, \boldsymbol{x}_{d_1}, \boldsymbol{x}_{d_2}, \boldsymbol{x}_{ans}) &= \frac{1}{2} D \quad (\boldsymbol{x}_{ref}, \boldsymbol{x}_{ans}) - \frac{1}{2} \sum_{i=1}^{2} D \quad (\boldsymbol{x}_{ref}, \boldsymbol{x}_{d_i}) \\ \text{where } D \quad (\boldsymbol{x}_1, \boldsymbol{x}_2) &= (||\boldsymbol{z}^{\boldsymbol{x}_1}_{scene} - \boldsymbol{z}^{\boldsymbol{x}_2}_{scene}||_2)^2 \text{ and } \boldsymbol{z}^{\boldsymbol{x}} = Encoder \quad (\boldsymbol{x}) \end{aligned} \tag{3.4}$$

The loss (c) is implemented via MINE (Belghazi et al., 2018). The strategy of MINE is to parameterize a variational formulation of the mutual information in terms of a neural network:

$$I_\Theta(z_s, z_v) = \sup_{\in\Theta} \mathbb{E}_{\mathbb{P}_{z_s z_v}}[T \quad] - \log(\mathbb{E}_{\mathbb{P}_{z_s} \otimes \mathbb{P}_{z_v}}[e^T \quad]). \tag{3.5}$$

This objective is optimized in an adversarial paradigm where T, the statistics network, plays the role of the critic and is fed with samples from the joint and marginal distribution. We added this loss to our pix2scene objective to minimize the mutual information estimate in both unsupervised and supervised training iterations. Once the model is trained, we answer 3D-IQTT questions, by inferring the latent 3D representation for each of the 4 images and we select the answer closest to the reference image as measured by L2 distance. We compared our model to two different baselines. The first one is composed of 4 ResNet-50 modules (He, Zhang, Ren, & Sun, 2016) with shared weights

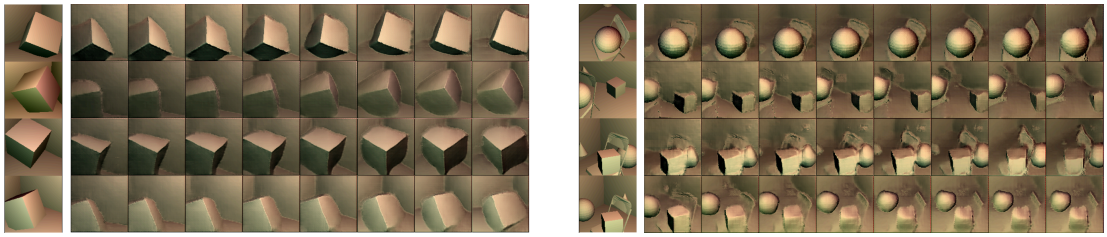


Figure 3.9: Representation analysis. Given a scene (first column), we rotate the camera around to visualize the model's understanding of 3D shape. As shown, the model correctly infers the unobserved geometry of the objects, demonstrating true 3D understanding of the scene. Figure 3.10 depicts the normal maps for the box setup. Videos of these reconstructions can be seen at https://bit.ly/2zADuqG.

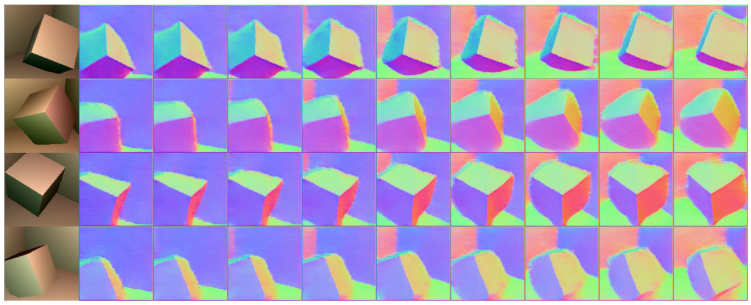


Figure 3.10: Normal views. Fixing representation and and smoothly changing camera view. For each row, the first column is the real input and other columns are the extrapolated views of that image from different views. Figure shows the corresponding normal maps of box inside a room setup in 3.9 for the extrapolations of box data (left) to stress the consistency.

followed by 3 fully-connected layers. We trained this CNN only on the labeled samples. Our second baseline has a similar architecture as the previous one but the fully-connected layers were removed. Instead of the supervised loss provided in the form of correct answers, it is trained on the contrastive loss (Koch, Zemel, & Salakhutdinov, 2015). This loss reduces the feature distance between the references and correct answers and maximizes the feature distance between the references and incorrect answers. A more detailed description of the networks and contrastive loss function can be found in the section 3.4.7.

Table 3.5 shows the 3D-IQTT results for our method and the baselines. The baselines were not able to interpret the underlying 3D structure of the data and its results are only slightly better than a random guess. The subpar performance of the Siamese CNN might be in part because the contrastive loss rewards similarities in pixel space and has no notion of 3D similarity. However, pix2scene achieved almost double these scores (0.6165) by leveraging the 3D knowledge of the ob-

(a) Input images **(b)** Reconstructed images **(c)** Reconstructed depth map

Figure 3.11: 3D IQ test task. Pix2scene reconstructions of the 3D-IQTT shapes.

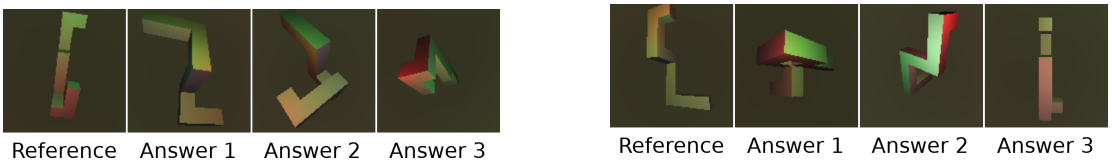


Figure 3.12: Sample questions from the 3D-IQ test task. For this "mental rotation" task, a set of reference images and 3 possible answers are presented. The goal is to find the rotated version of the reference 3D model. To solve this task, the human or the model has to infer the 3D shape of the reference from the 2D image and compare that to the inferred 3D shapes of the answers. The correct answers to these two examples are in the footnote.

jects for solving the task.

Labeled Samples	CNN	Siamese CNN	Pix2Scene (Ours)
1,000	0.3392	0.3701	0.5789
10,000	0.3649	0.3752	0.6165

Table 3.5: 3D-IQTT quantitative results. The test accuracy of the 3D-IQ test task show that the CNN baselines struggle to solve this task Pix2scene is able to understand the underlying 3D structure of the images and solve the task

3.4.7 Architecture for 3D IQTT Evaluations

Pixel2Scene architecture remains similar to the ones in previous sections but with higher capacity on decoder and critic as this task is more challenging and complex. The more important difference is

[1] three [2] two

that for those experiments we do not condition the networks with the camera pose to be fair with the baselines. In addition to the three networks we have a statistics network (see table 3.6) that estimates and minimizes the mutual information between the two set of dimensions in the latent code using MINE(Belghazi et al., 2018). Out of 128 dimensions for z we use first 118 dimensions for represent scene-based information and rest to encode view based info.

Statistics Network

Layer	Outputs	Kernel size	Stride	BatchNorm	Activation
Input $[z[:118], z[118:]]$	$1 \times 1 \times 128$				
Convolution	$1 \times 1 \times 256$	1×1	1	No	ELU
Convolution	$1 \times 1 \times 512$	1×1	1	No	ELU
Convolution	$1 \times 1 \times 1$	1×1	2	No	None

Table 3.6: Pix2scene statistics network architecture.

The architecture of the baseline networks is shown in figure 3.13. The contrastive loss using for training this baselines is shown in figure 3.6.

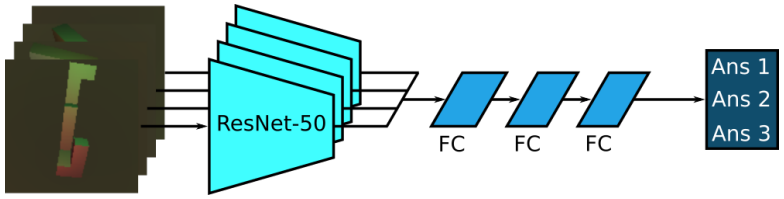


Figure 3.13: 3D-IQTT baseline architecture. The ResNet-50 all share the same weights and were slightly modified to support our image size. "FC" stands for fully-connected layer and the hidden node sizes are 2048, 512, and 256 respectively. The output of the network is encoded as one-hot vector.

The contrastive loss from equation 3.6 is applied to the 2048 features that are generated by each ResNet block. $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the input images, y is either 0 (if the inputs are supposed to be the same) or 1 (if the images are supposed to be different), G is each ResNet block, parameterized by ϑ, and m is the margin, which we set to 2.0. The loss function is from Hadsell, Chopra, and LeCun, 2006

but used slightly differently.

$$\begin{aligned}\mathcal{L}(\boldsymbol{x}_1, \boldsymbol{x}_2, y) &= (1-y)\frac{1}{2}(D \quad (\boldsymbol{x}_1, \boldsymbol{x}_2))^2 + (y)\frac{1}{2}(max(0, m - D \quad (\boldsymbol{x}_1, \boldsymbol{x}_2)))^2 \\ D \quad (\boldsymbol{x}_1, \boldsymbol{x}_2) &= ||G \quad (\boldsymbol{x}_1) - G \quad (\boldsymbol{x}_2)||_2\end{aligned} \tag{3.6}$$

3.5 Discussion

In this paper we proposed a generative approach to learn 3D structural properties from still images. To achieve this, we leveraged the generative capability and efficient inference pathway provided by the adversarial bi-directional models. We showcased the model's ability to learn true representations of the 3D scenes from image pixels through a variety of experiments from 3D primitives to challenging scenes. In turn, this learned representation allowed us to synthesize novel scene configurations. We showcased our generative capabilities on the ShapeNet dataset. Lastly, we provided quantitative evidence that support our argument by introducing a novel IQ-task in a semi-supervised setup.

4

The Good, the Bad and the Cheap Robots

In this chapter we discuss some of the remaining issues in our contributions and conclude with a summary of currently ongoing and future research.

4.1 Discussion

We set out to create new methods to train low-cost robots. In chapter 1 we assumed we have all components in place, had a simulator and a sophisticated perception method, and only needed a way of transferring our policy. We expected that by learning a model of the dynamics of the real robot we could augment the simulation, so that policies learned in simulation would transfer better to the real robot. We demonstrated that this is true for some sim2sim environments and for one task on an actual robot. However, we didn't yet investigate the extrapolation behavior of our augmented simulator over time. Our longest-running rollouts were around 20s long. While the LSTM model presented in that work was able to compensate for such time spans, we speculate that after some longer runtime the error would compound increasingly to a point where the augmented simulator is actually not beneficial anymore. There is also a chance that the LSTM compensates for indeterminate runtime if the executed motions aren't monotonic. We noticed this several times during experimentation; as long as the joint was trying to reach a certain target configuration, the error was accumulating. Nonetheless, as soon as the direction of joint movement was inverted, the error vanished. This is anecdotal, however, and requires more investigation. Another useful analysis would

be to verify if the method works on very high-dimensional problems such as visual servoing, e.g., where the robot has to pick up an object by observing itself in an image or video stream.

In chapter 2 we stripped away one layer of our robot learning stack and looked at new simulators. We found that there was a lack of simulators where the dynamics could easily be varied, and a lack of multimodal simulations for language-based robot training. We addressed this issue by contributing several simulators. There are, however, some outstanding problems and some new ones. The main problem in the Duckietown- and Ergo Jr.-based simulations can be seen in their adaptation or lack thereof. Both are geared towards low-cost robotic platforms, but not every robotics research group is interested in compensating for the additional difficulties that these platforms burden themselves with like backlash, strongly heat-varying control, and general control noise. Some robotics research is financed with concrete application scenarios in mind and while the low-cost robots are useful for new methods and proof-of-concept, high-precision robots might be required for deliverables. Even if a group is specifically looking for relatively low-cost robots, both the Duckiebot and the Poppy Ergo Jr. might appear too limiting compared to low/mid-range solutions like the Turtlebot 3 and the Dobot Magician, respectively. However, as mentioned in the introduction, the field of robot learning is continuing to grow and therefore, the need for new methods and low-cost robotic platforms is only rising.

For the HMRI dataset we see a large drawback in the current lack of suitable machine learning models that support such a complex lifelong learning task. Therefore, it is hard to quantify the quality of the dataset other than the naive baseline CNN approach, which can't be trained end-to-end due to the size of the dataset. This might change in the future, of course, and it might also be interesting to investigate this further through zero/few-shot transfer and automatic pixel-level object proposals. In the case of the HoME environment, the *new* issue is that after our inital release, several new environments were released that all attempt to supersede our environment: Savva, Chang, Dosovitskiy, Funkhouser, and Koltun (2017), Martinez-Gonzalez et al. (2018), K.-T. Lai, Lin, Kang, Liao, and Chen (2018), Xia et al. (2018), Puig et al. (2018), Yan et al. (2018). We assume this indicates that there is indeed a need for increasingly complex simulations. Most of these releases contain a similar feature set to ours but improve through more photorealism or more interactivity. Despite some of them featuring semantic segmentations, none combine them into object relations or descriptions. We expect future agents to have a stronger language aspect, which is why we assume that this compo-

http://www.robotis.us/turtlebot-3/

https://www.dobot.cc/dobot-magician/product-overview.html

Puig et al. (2018) is somewhat of an exception here in that it allows the user to create action scripts that are then carried out by an animated avatar in the simulation for data generation.

nent will be crucial for other upcoming platforms.

In the case of the Pix2Scene project, there are several outstanding issues: Most of our experiments revolved around monochromatic objects in the scene, colored point lights, no textures, and no reflections. We did, however, conclude preliminary experiments with vivid object colors and monochromatic lights. This is still far from real environments where the lighting is much more nuanced and hard to decompose, where, for example, a bright red wall changes the perceived color of nearby objects, and where objects aren't as smooth as in our simple scenes. This method might require more perception priors to make better guesses about scene objects; or an additional scene decomposition step, attempting to remove textures or even colors from objects while preserving shading. Additionally, we experimented mostly with 128 x 128 resolution images. One might assume that this is vastly too low, but a contemporary LIDAR, like the VLS-128 only has an effective resolution of roughly 3272 x 363. Our contribution could achieve similar results by doubling the x/y resolution, rotating the camera and merging the resulting scans.

4.2 Current and Future Research

None of the different research projects in this work were carried out in isolation. As a result, the projects discussed in this work continue beyond the scope of this thesis. Currently, the following are being actively developed:

- For the sim2real contribution in chapter 1 more challenging physical experiments are being developed. The approach was evaluated with only a single real robot experiment with 6 DoF and a static target, which is why we are currently exploring a "sword-fighting" task. This has a similar setup as the sparring match, but where both robots are equipped with pencils and have the goal of hitting each other. This is achieved with a self-play approach as discussed in Silver et al. (2017), which pits the current policy on one of the robot against a collection of previous policies on the other robot. This task is designed as "stress test" for our method since both robots are moving at high speeds to avoid getting hit. On top of this, we are also experimenting with the active goal babbling as initial exploration policy, as mentioned in 1.2, and what the effects of this are on the transfer policy.
- For the Duckietown project from chapter 2.2 we are preparing for the upcoming NIPS and ICRA challenges. This entails disseminating the basics of reinforcement learning in simulation to the audience, as well as work on the submission and sim2real transfer procedure. We gave our audience a wide array of submission possibilities, from a traditional vision pipeline in ROS, to an imitation learning baseline in TensorFlow. All of these still have to be extensively tested for their transferability to and performance on the real robot.

https://www.velodynelidar.com/vls-128.html

- For the Pix2Scene project from chapter 3 we are currently extending the model to include dedicated color and shadow understanding and prediction components. We are also developing a web interface for the IQ test task that was introduced in that work in order to get a human performance baseline. In addition to this, we are preparing for a workshop for CVPR 2019, that contains a challenge revolving around the IQ test task.

Beyond our currently ongoing work, we also have some ideas on which future directions might be interesting for our projects:

- For the Neural-Augmented Simulator approach, we would like to implement a new image transfer method in addition to our dynamics transfer. There have been similar ideas in Pecka, Zimmermann, Petrlík, and Svoboda (2018), Stein and Roy (2018), Yang, Liang, Wang, and Xing (2018) but all these approaches are highly task/policy-specific. These approaches are not likely to transfer well to domains where either the sampling policy can't be rolled out without risking damage to the physical system, or where matching samples from both simulation and reality are hard to construct. For such a system to work, a model would need to be trained on unpaired samples, while retaining enough state information about both environments.

- For Duckietown, we are considering adding more complexity in terms of sensors and actuators to the robot. There have been experiments with inertial measurement units (IMUs) before, but we might also explore wheel encoders. Since the Duckiebot is already controlled with a Raspberry Pi, it should be possible to mount a Poppy Ergo Jr. arm on the Duckiebot. From the hardware perspective, this would only require a higher-powered battery and step-down DC transformer. From the software perspective, this would require a ROS interface to the Ergo Jr., or conversely, a python interface to the Duckiebot.

- After the work on the Household Multimodal Environment, we plan to work on training embodied agents to interpret and enact natural language instructions. Since there are now so many new environments, many of which are more detailed than ours, we are considering extending one of them (like GibsonEnv by Xia et al., 2018) with our semantic annotation engine and experimenting with language learning curricula. This would consist of increasingly difficult instructions paired with an increasingly wider range of actions in the 3D environments.

- When the Pix2Scene model has been extended to realistic color and shadows, we would like to evaluate our work on full indoor scenes. For a quantitative evaluation of such complex environments, ground truth would be required, which could be provided by a house simulator, like the aforementioned GibsonEnv. We are also considering a reconstruction loop that involves 3D-printing scanned objects and using the feedback to improve the recognition process.

This concludes the conclusion. We hope to have contributed to current and future research on this problem by developing new tools for sim2real transfer, providing more sophisticated and detailed new simulation environments, and providing new ways of perceiving 3D environments.

A

Summary (French)

Cette thèse est soutenue par le projet européen “CHIST-ERA” sur la “compréhension linguistique interactive” (IGLU). L’objectif de ce projet est de créer un système robotique pouvant être utilisé comme aide domestique. Cet assistant interagirait de façon naturelle avec l’opérateur humain par le biais de la parole et serait en mesure d’acquérir de nouvelles compétences motrices et de nouveaux mots dans le cadre de son utilisation. Nous avons travaillé à l’envers à partir de cette idée. Nous avons supposé que nous avions un robot capable auquel nous n’avions besoin que de fournir le cerveau. Pour des projets à grande échelle comme celui-ci, il est généralement plus facile de réaliser toutes les expériences en simulation.

Le projet IGLU a plusieurs étapes et 7 universités différentes y ont contribué. L’INRIA avait pour tâche principale de s’assurer que la politique apprise en simulation par les autres partenaires fonctionnerait aussi dans le monde réel. En effet, les politiques uniquement entraînées en simulation ne fonctionnent pas aussi bien sur le robot réel. Ce phénomène est communément appelé «reality gap». La contribution principale de ce travail est un nouvel algorithme qui modifie les simulations de manière à ce que les politiques qui y sont entraînées fonctionnent sur le robot réel sans ajustement (voir le chapitre 1.1).

Il existe plusieurs manières d’aborder le problème. Bongard and Lipson (2004), Zagal and Ruiz-Del-Solar (2007), Koos, Mouret, and Doncieux (2013) sont dans la catégorie des approches évolutives. Ceux-ci utilisent un simulateur qui évolue avec la politique ou bien font de la transférabilité une partie intégrante de la fonction d’aptitude. Ainsi, ce n’est pas la politique la plus per-

formante qui est choisie, mais plutôt celle qui fonctionne de la manière la plus fiable sur le système physique. Le grand inconvénient de ces approches est qu'elles sont «on-policy», c'est-à-dire qu'elles ont besoin que la politique soit évaluée pendant l'entraînement. Par conséquent, le robot risque de s'endommager lui-même ou d'autres personnes, en particulier lors des premières séances d'entraînement. Elles sont également difficiles à mettre en œuvre si la tâche est dynamique, comme le lancer d'une balle, car après chaque essai, un humain doit réinitialiser l'environnement (c'est-à-dire ramasser la balle et la remettre en place). De plus, nous n'avons trouvé aucune approche fonctionnant avec une grande dimensionnalité d'état, comme c'est le cas avec le contrôle moteur visuel. Il existe une autre famille d'approches, principalement issues du groupe de Pieter Abbeel, (Peng et al., 2017; Joshua Tobin, Zaremba, & Abbeel, 2017; Josh Tobin et al., 2017), qui relèvent toutes de la catégorie de la randomisation de domaine. L'idée principale de cette méthode se résume comme suit:

> With enough variability in the simulator, the real world may appear to the model as just another variation. [Avec suffisamment de variabilité dans le simulateur, le monde réel peut apparaître au modèle comme une autre variation.] (Josh Tobin et al., 2017)

Bien que cette approche permette un transfert instantané, elle est plutôt inefficace en termes d'échantillonnage. Cela nécessite également que l'opérateur ait sélectionné les bonnes dimensions à randomiser et dans la plage correcte.

Nous présentons une nouvelle approche qui détecte la différence entre le simulateur et le robot réel indépendamment de la tâche et l'applique à la simulation. Le simulateur modifié peut être utilisé pour former des politiques qui se transfèrent nettement mieux que celles apprises dans le simulateur non modifié. La mise en œuvre de cette approche est la suivante:

1. Exécuter des mouvements aléatoires sur les vrais robots et enregistrer l'état pour N pas de temps.
2. Exécuter les mêmes actions avec le même Δt en simulation. À chaque pas, enregistrer l'écart entre l'état du simulateur et l'état réel. Après chaque pas, régler l'état du simulateur sur celui du robot réel avant d'exécuter l'action suivante.
3. Entraîner un LSTM à prédire cet écart.
4. Chaque fois que vous entraînez une politique sur ce robot, régler le simulateur sur l'état compensé par les différences à chaque pas avant d'exposer l'état à l'algorithme d'apprentissage de la stratégie.

Nous avons évalué cette méthode sur 3 environnements sim2sim et 1 environnement sim2real. Les environnements sim2sim consistaient chacun en une version «réelle» dans laquelle nous avons

ajouté un backlash à un ou plusieurs joints, et un environnement «simulé», qui n'avait pas de backlash. Nous avons choisi l'option backlash, car nous avons constaté que les différences de frottement, de masse et de couple sont beaucoup plus faciles à modéliser et ont un impact moindre sur la politique après le transfert. Pour les environnements sim2real, nous avons mis en place une tâche de "combat d'entraînement" dans laquelle un robot défenseur avec un bouclier se déplace à des positions aléatoires et un robot attaquant avec une «épée» tente de toucher le bouclier aussi vite que possible. Dans tous ces environnements, nous avons étudié comment la différence de performance entre (1) la politique experte, qui a été formée directement sur l'environnement cible / réel, (2) la politique de simulation, qui a été entraînée en simulation et transférée sans ajustement, (3) la politique de prédiction, dans laquelle aucun simulateur n'a été utilisé et l'état suivant était directement prédit par le réseau de neurones, et (4) la politique de transfert, qui utilisait notre simulateur augmenté.

Nous avons constaté que pour 3 de nos 4 environnements, ce qui suit était vrai: la politique experte a donné les meilleurs résultats, la politique de simulation a mal fonctionné, la politique de prédiction a obtenu de meilleurs résultats que la stratégie de simulation, et la politique de transfert a obtenu de meilleurs résultats que la stratégie de prédiction, généralement égaux ou légèrement inférieurs à ceux de la stratégie experte. La seule exception à cette règle est l'un des environnements sim2sim dans lesquels le robot devait frapper une balle. Notre approche a réussi à frapper la balle dans la bonne direction, mais pas avec suffisamment de force, ce qui a entraîné une performance légèrement inférieure à celle de la politique de prédiction. Ces résultats étaient positifs et nous voudrions dans l'avenir pouvoir évaluer cette méthode sur des environnements plus différents ainsi que l'impact de la couverture de l'espace d'état initial sur la simulation augmentée résultante.

Pour évaluer cette méthode, nous avons développé un ensemble de nouveaux environnements en simulation et modifie les environnements existants pour permettre une commutation rapide des dynamiques. Ces derniers constituent les contributions logicielles de cette thèse et pourront être réutilisés dans d'autres projets de recherche. Vous trouverez des informations complémentaires au chapitre 2.1.

Au cours de cette thèse, nous nous sommes également impliqués dans le projet Duckietown, une plate-forme robotique peu coûteuse et de taille reduite en forme de voiture. Ces Duckiebots sont utilisés pour enseigner aux étudiants des cycles supérieurs les voitures autonomes et la robotique. Pour les conférences NIPS 2018 et ICRA 2019, nous avons fourni l'architecture de simulation qui permet aux participants de développer leurs propres algorithmes de conduite en simulation et de les évaluer automatiquement sur de vrais robots. Nous avons également proposé une méthode de

Le backlash est une imperfection des boîtes d'engrenages où l'un des deux engrenages imbriqués peut légèrement se déplacer sans l'autre.

référence basé sur l'apprentissage par renforcement, qui montre comment l'apprentissage complet des politiques peut être utilisé pour apprendre à un robot à conduire (voir 2.2).

Après le détour par Duckietown, nous avons poursuivi au chapitre 2.3 sur le projet IGLU. Avec l'Université de Saragosse, nous avons développé un nouveau benchmark pour tester l'acquisition du langage dans les robots, appelé référentiel d'interaction multimodal homme-robot (MHRI). Pour cet ensemble de tests, nous avons enregistré plusieurs heures avec un microphone et plusieurs caméras RGB-D capturant des êtres humains en interaction avec un robot, montrant et pointant des objets domestiques courants tout en les nommant. Cet ensemble de données peut être utilisé pour valider l'acquisition continue de nouvelles connaissances dans les systèmes robotiques déployés dans le monde réel. Celui-ci est unique (1) par sa difficulté et (2) par le fait qu'il fut enregistré (principalement) via les capteurs embarqués du robot. Les participants humains n'avais pas reçu d'instruction sur ce qu'ils devaient dire, ce qui a entraîné des interactions naturelles mais parfois bruyantes. Une autre partie de cette même contribution est un système de reconnaissance d'objets base sur ces données multimodales qui ne repose pas sur l'apprentissage profond.

Nous avons constaté qu'il n'existait pas d'environnements de simulation dans lesquels un agent pouvait être formé pour acquérir les compétences nécessaires pour. Par conséquent, nous avons développé un nouvel environnement de simulation appelé "Environnement domestique multimodal" dans lequel les robots peuvent acquérir des compétences linguistiques et de navigation grâce à de multiples entrées sensorielles (voir le chapitre 2.4 pour plus de détails). Le simulateur consiste en un système modulaire basé sur le moteur de jeu Pandas. Nous avons inclus des modèles de maisons et d'appartements du jeu de données SUNCG et, en les chargeant dans le moteur de jeu, permettons aux agents virtuels de parcourir les salles et d'interagir avec les objets. Nous avons fourni les modules suivants: (1) moteur de rendu - pour le rendu des salles SUNCG, (2) moteur physique - pour autoriser des collisions ainsi que des interactions simples comme ouvrir / fermer des portes et ramasser des objets, (3) moteur acoustique - pour permettre au son de voyager dans les environnements, de rebondir sur les murs et d'être absorbé par des matériaux tendres pour créer une expérience acoustique authentique, (4) moteur sémantique - qui peut annoter une scène donnée et décrire des objets par leurs attributs ou en relation avec d'autres objets et salles.

Nous avons également considéré que malgré nos efforts pour créer des politiques qui généralisent bien et des environnements d'entraînement réalistes et proches de celui où le robot sera déployé, il est possible que le robot se trouve dans un environnement qui ne fait pas partie de la simulation. Pour faire face à une telle situation, l'agent devrait construire un modèle 3D de la scène. Il est possible que l'agent soit équipé d'un capteur de profondeur capable de capturer l'intégralité de la scène à partir d'un seul balayage, mais nous avons envisagé le cas où seule une caméra 2D est disponible

car elles sont moins chères et plus courantes. L'avantage supplémentaire de cette approche est que la méthode peut être utilisée non seulement pour créer des mondes 3D sur un robot déployé, mais également pour bootstrapper rapidement de nouveaux environnements simulés avec seulement une caméra smartphone. Le chapitre 3 détaille cette contribution. Il existe plusieurs approches existantes (Soltani et al., 2017; J. Wu, Wang, et al., 2016; J. Wu, Xue, et al., 2016), mais elles reposent soit sur une représentation voxel, soit sur une représentation maillée. Les représentations en maillage ont du mal à être mises à l'échelle car il n'y a pas de correspondance directe entre les pixels de l'image source et les maillages nécessaires pour modéliser l'objet. De plus, pour chaque nouvelle surface, 12 valeurs supplémentaires sont requises (les 3 ensembles de 3 coordonnées des sommets et au moins le vecteur normal). Pour les voxels, le problème principal réside dans le fait que le modèle doit faire une hypothèse sur le contenu des objets 3D. Dans l'approche que nous proposons, nous utilisons la représentation exotique "splat / surfel" qui transforme chaque pixel en un point 3D comme un nuage de points, mais aussi, crée une surface 3D continue pouvant être utilisée dans un simulateur. Nous avons implémenté cela avec un réseau antagoniste génératif bidirectionnel qui prédit, à partir d'une image, la profondeur de chaque pixel en fonction de l'ombrage, après quoi nous déduisons l'orientation (vecteur normal). Cette représentation intermédiaire est alimentée via un moteur de rendu surfel que nous avons construit qui est différentiable et nous permet de calculer directement la perte de l'image reconstruite par rapport à l'image d'entrée. La bidirectionnalité permet un entraînement plus stable et prédit l'image d'entrée à partir d'une représentation latente donnée. Nous avons validé cette méthode dans deux contextes: (1) qualitativement en reconstruisant des pièces avec plusieurs objets à l'intérieur et en comparant visuellement les images d'entrée avec les reconstructions et les modèles 3D et (2) quantitativement avec une tâche de test de QI que nous avons créée. Pour celle-ci, nous avons créé des modèles 3D similaires à ceux utilisés dans les tests de QI dans les tâches de rotation mentale. Chaque question du test contient une image de référence et 3 réponses possibles, dont l'un est l'objet de référence, mais tourné de manière aléatoire. L'ensemble de données contient un petit sous-ensemble d'échantillons d'apprentissage étiquetés (où la réponse correcte est connue) et un grand corpus non étiqueté. Notre modèle a été pré-entraîné pour reconstruire les données non étiquetées. Il utilise la distance euclidienne dans l'espace latent pour prédire quelle réponse est la plus proche de la référence. Il a été affiné sur les échantillons étiquetés toutes les 1 000 étapes d'apprentissage. Comme modèles de référence, qu'on utilisera pour les comparaisons, nous avons implémenté un CNN qui prédit le softmax sur les réponses, en prenant en entrée les 4 images, et un réseau siamois entraîné pour créer une seule représentation latente de l'image de référence et la bonne réponse et pour répondre à une question afin de sélectionner la réponse avec la plus petite distance des représentations latentes. Dans l'expérience (1), nous avons constaté que nos images re-

construites semblaient convaincantes, bien que légèrement plus lisses en raison de l'estimation du vecteur normal. Pour l'expérience (2), notre implémentation a obtenu un score presque deux fois supérieur à celui des modèles de référence. Comme nous n'avons pas trouvé d'autres mesures pour la reconstruction 3D autres que la distance de Hausdorff, le jeu de données de test de QI a été rendu public, de sorte que d'autres approches de reconstruction 3D puissent également évaluer quantitativement leur approche.

Le projet IGLU était très ambitieux depuis le début et vient d'être prolongé pour un an. Jusqu'à présent, il n'y a pas de solution au problème du robot assistant domestique. En développant de nouveaux outils pour le transfert sim2real, en fournissant de nouveaux environnements de simulation plus sophistiqués et détaillés et en offrant de nouvelles façons de percevoir les environnements 3D, nous espérons avoir contribué aux recherches actuelles et futures sur ce problème.

Bibliography

Abdullah, L. N. & Noah, S. A. M. (2008). Integrating audio visual data for human action detection. In *Int. conf. computer graphics, imaging and visualisation* (pp. 242–246). IEEE.

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., & Süsstrunk, S. (2012). SLIC superpixels compared to state-of-the-art superpixel methods. *Trans. on Pattern Analysis and Machine Intelligence*, *34*(11), 2274–2282.

Adomavicius, G. & Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook* (pp. 217–253).

Anderson, J., Baltes, J., & Cheng, C. T. (2011). Robotics competitions as benchmarks for AI research. *The Knowledge Engineering Review*, *26*(1), 11–17. doi:10.1017/S0269888910000354

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., ... Hengel, A. v. d. (2017). Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. *arXiv preprint arXiv:1711.07280*.

Anonymous. (2018). Building generalizable agents with a realistic and rich 3d environment. *Under Submission at ICLR*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., & Parikh, D. (2015). Vqa: Visual question answering. In *Cvpr*.

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein generative adversarial networks. *International Conference on Machine Learning (ICML)*.

Azagra, P., Golemo, F., Mollard, Y., Lopes, M., Civera, J., & Murillo, A. C. (2017). A multimodal dataset for object model learning from natural human-robot interaction. *Intelligent Robots and Systems (IROS) 2017 IEEE/RSJ International Conference*.

Baltes, J., Tu, K.-Y., Sadeghnejad, S., & Anderson, J. (2017). HuroCup: competition for multi-event humanoid robot athletes. *The Knowledge Engineering Review*, *32*.

Baranes, A. & Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, *61*(1), 49–73.

Barsalou, L. W. (2008). Grounded cognition. *Annual Review of Psychology*, *59*, 617–645.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., ..., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Behnke, S. (2006). Robot competitions-ideal benchmarks for robotics research. In *Proc. of iros-2006 workshop on benchmarks in robotics research*. Institute of Electrical and Electronics Engineers (IEEE).

Belghazi, M. I., Baratin, A., Rajeshwar, S., Ozair, S., Bengio, Y., Hjelm, D., & Courville, A. (2018). Mutual information neural estimation. In *International conference on machine learning*.

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *JAIR*, *47*, 253–279.

Bojar, O., Chatterjee, R., Federmann, C., Haddow, B., Huck, M., Hokamp, C., ... Turchi, M. (2015). Findings of the 2015 workshop on statistical machine translation. In *Workshop on statistical machine translation*.

Bongard, J. & Lipson, H. (2004). Once more unto the breach: Co-evolving a robot and its simulator. In *Proceedings of the ninth international conference on the simulation and synthesis of living systems (alife9)* (pp. 57–62).

Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., ..., Konolige, K., et al. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *Robotics and automation (icra), 2018 ieee international conference on*. IEEE.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Brodeur, S., Perez, E., Anand, A., Golemo, F., Celotti, L., Strub, F., ... Courville, A. (2018). Home: A household multimodal environment. *6th International Conference on Learning Representations (ICLR) 2018, Workshop Track*.

Censi, A., Paull, L., Tani, J., Zilly, J., Ackermann, T., Beijbom, O., ... Frazzoli, E. (2018). The ai driving olympics at nips 2018. *Robotics: Science and Systems (RSS) 2018, Workshop Track*.

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., ... Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. *arXiv*.

Chang, A., Dai, A., Funkhouser, T., Halber, M., Niessner, M., Savva, M., ... Zhang, Y. (2017). Matterport3d: Learning from rgb-d data in indoor environments. *3DV*.

Chaplot, D. S., Parisotto, E., & Salakhutdinov, R. (2018). Active neural localization. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=ry6-G_66b

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., & Salakhutdinov, R. (2017). Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*.

Chaudhuri, S., Kalogerakis, E., Guibas, L., & Koltun, V. (2011). Probabilistic reasoning for assembly-based 3d modeling. In *Acm siggraph*.

Choy, C. B., Xu, D., Gwak, J., Chen, K., & Savarese, S. (2016). 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. *ArXiv*.

Collet, A., Berenson, D., Srinivasa, S. S., & Ferguson, D. (2009). Object recognition and full pose registration from a single image for robotic manipulation. In *Ieee int. conf. on robotics and automation* (pp. 48–55). doi:10.1109/ROBOT.2009.5152739

Dalal, N. & Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Computer vision and pattern recognition* (Vol. 1, pp. 886–893). IEEE.

de Vries, H., Strub, F., Chandar, S., Pietquin, O., Larochelle, H., & C. Courville, A. (2017). Guesswhat?! visual object discovery through multi-modal dialogue. In *Cvpr*.

Deisenroth, M. & Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th international conference on machine learning (icml-11)* (pp. 465–472).

Dhall, A., Ramana Murthy, O., Goecke, R., Joshi, J., & Gedeon, T. (2015). Video and image based emotion recognition challenges in the wild: Emotiw 2015. In *International conference on multimodal interaction*.

Donahue, J., Krähenbühl, P., & Darrell, T. (2016). Adversarial feature learning. *arXiv preprint arXiv:1605.09782*.

Dumoulin, V., Belghazi, I., Poole, B., Lamb, A., Arjovsky, M., Mastropietro, O., & Courville, A. (2016). Adversarially learned inference. *arXiv preprint arXiv:1606.00704*.

Finn, C., Levine, S., & Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning* (pp. 49–58).

Fisher, K., Hirsh-Pasek, K., & Golinkoff, R. M. (2012). Fostering mathematical thinking through playful learning. *Contemporary debates on child development and education*, 81–92.

Fisher, M., Ritchie, D., Savva, M., Funkhouser, T., & Hanrahan, P. (2012). Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics*.

Fu, J., Levine, S., & Abbeel, P. (2016). 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* (pp. 4019–4026). IEEE.

Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., ... Lempitsky, V. (2016). Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, *17*(59), 1–35.

Gauthier, J. & Mordatch, I. (2016). A paradigm for situated and goal-driven language learning. *NIPS Machine Intelligence Workshop*.

Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer vision and pattern recognition (cvpr), 2012 ieee conference on* (pp. 3354–3361). IEEE.

Golemo, F., Ali Taiga, A. A., Oudeyer, P.-Y., & Courville, A. (2018). Sim-to-real transfer with neural-augmented robot simulation. *Conference on Robot Learning (CoRL)*.

Gong, W., Gonzàlez, J., Tavares, J. M. R. S., & Roca, F. X. (2012). A new image dataset on human interactions. In *Articulated motion and deformable objects: 7th international conference, amdo 2012, port d'andratx, mallorca, spain, july 11-13, 2012. proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... Bengio, Y. (2014). Generative adversarial nets. *Advances in Neural Information Processing Systems (NIPS)*.

Goslin, M. & Mine, M. R. (2004). The panda3d graphics engine. *Computer*, *37*(10), 112–114.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., & Courville, A. C. (2017). Improved training of wasserstein gans. *Advances in Neural Information Processing Systems (NIPS)*.

Hadsell, R., Chopra, S., & LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *Null* (pp. 1735–1742). IEEE.

Handa, A., Pătrăucean, V., Stent, S., & Cipolla, R. (2016). Scenenet: An annotated model generator for indoor scene understanding. In *Icra*.

Hanna, J. & Stone, P. (2017). Grounded action transformation for robot learning in simulation. In *Proceedings of the 31st aaai conference on artificial intelligence (aaai)*. San Francisco, CA.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., ..., Teplyashin, D., et al. (2017). Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*.

Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., ... Lerchner, A. (2017). Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*.

Higuera, J. C. G., Meger, D., & Dudek, G. (2017). Adapting learned robotics behaviours through policy adjustment. In *Robotics and automation (icra), 2017 ieee international conference on* (pp. 5837–5843). IEEE.

Hochreiter, S. & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hua, B.-S., Pham, Q.-H., Nguyen, D. T., Tran, M.-K., Yu, L.-F., & Yeung, S.-K. (2016). Scenenn: A scene meshes dataset with annotations. In *3dv* (pp. 92–101). IEEE.

Huang, S., Qi, S., Zhu, Y., Xiao, Y., Xu, Y., & Zhu, S.-C. (2018). Holistic 3D Scene Parsing and Reconstruction from a Single RGB Image. *arXiv*.

Iagnemma, K. & Overholt, J. (2015). Introduction. *Journal of Field Robotics*, *32*(2), 187–188.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).

Iravani, P., Hall, P., Beale, D., Charron, C., & Hicks, Y. (2011). Visual object classification by robots, using on-line, self-supervised learning. In *Ieee int. conf. on computer vision workshops* (pp. 1092–1099).

Izadinia, H., Shan, Q., & Seitz, S. (2017). IM2CAD. *arXiv*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *European conference on artificial life* (pp. 704–720). Springer.

James, S. & Johns, E. (2016). 3d simulation for robot arm control with deep q-learning. *arXiv preprint arXiv:1609.03759*.

Johnson, M., Hofmann, K., Hutton, T., & Bignell, D. (2016). The malmo platform for artificial intelligence experimentation. In *Ijcai*.

Kalogerakis, E., Chaudhuri, S., Koller, D., & Koltun, V. (2012). A probabilistic model for component-based shape synthesis. *ACM Transactions in Graphics*, *31*(4), 55:1–55:11.

Kaplan, A. (2017). *The conduct of inquiry: Methodology for behavioural science*. Routledge.

Kapur, M. (2008). Productive failure. *Cognition and instruction*, *26*(3), 379–424.

Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation. *International Conference on Learning Representations (ICLR)*.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016). ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *Computational intelligence and games*.

Kenney, J., Buckley, T., & Brock, O. (2009). Interactive segmentation for manipulation in unstructured environments. In *Ieee int. conf. on robotics and automation* (pp. 1377–1382).

Kidziński, Ł., Mohanty, S. P., Ong, C., Hicks, J. L., Carroll, S. F., Levine, S., ... Delp, S. L. (2018). Learning to Run challenge: Synthesizing physiologically accurate motion using deep reinforcement learning. *ArXiv e-prints*.

Kiela, D., Bulat, L., Vero, A. L., & Clark, S. (2016). Virtual embodiment: A scalable long-term strategy for artificial intelligence research. In *Machine intelligence workshop at nips*.

Kingma, D. P. [Diederik P] & Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*.

Kingma, D. P. [Diederik P.] & Welling, M. (2014). Auto-encoding variational bayes. *International Conference on Learning Representations (ICLR)*.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997). Robocup: The robot world cup initiative. In *Proceedings of the first international conference on autonomous agents* (pp. 340–347). ACM.

Kobbelt, L. & Botsch, M. (2004). A survey of point-based techniques in computer graphics. *Computers & Graphics*, *28*(6), 801–814.

Koch, G., Zemel, R., & Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *Icml deep learning workshop* (Vol. 2).

Koos, S., Mouret, J.-B., & Doncieux, S. (2010). Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on genetic and evolutionary computation* (pp. 119–126). ACM.

Koos, S., Mouret, J.-B., & Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*, *17*(1), 122–145.

Krainin, M., Curless, B., & Fox, D. (2011). Autonomous generation of complete 3D object models using next best view manipulation planning. In *Ieee int. conf. on robotics and automation* (pp. 5031–5037).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012a). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012b). Imagenet classification with deep convolutional neural networks. In *Nips*.

Kulkarni, T. D., Whitney, W., Kohli, P., & Tenenbaum, J. B. (2015). Deep Convolutional Inverse Graphics Network. *Advances in Neural Information Processing Systems (NIPS)*.

Lai, K., Bo, L., Ren, X., & Fox, D. (2011). A large-scale hierarchical multi-view RGB-D object dataset.

Lai, K.-T., Lin, C.-C., Kang, C.-Y., Liao, M.-E., & Chen, M.-S. (2018). Vivid: Virtual environment for visual deep learning. In *2018 acm multimedia conference on multimedia conference* (pp. 1356–1359). ACM.

Laine, S., Siltanen, S., Lokki, T., & Savioja, L. (2009). Accelerated beam tracing algorithm. *Applied Acoustics*, *70*(1), 172–181.

Landau, B., Smith, L., & Jones, S. (1998). Object perception and object naming in early development. *Trends in cognitive sciences*, *2*(1), 19–24.

Lapeyre, M., Rouanet, P., Grizou, J., Nguyen, S., Depraetre, F., Le Falher, A., & Oudeyer, P.-Y. (2014). Poppy project: Open-source fabrication of 3d printed humanoid robot for science, education and art. In *Digital intelligence 2014* (p. 6).

Lee, C., Badrinarayanan, V., Malisiewicz, T., & Rabinovich, A. (2017). Roomnet: End-to-end room layout estimation. *CoRR*, *abs/1703.06241*.

Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., & Carin, L. (2017). Alice: Towards understanding adversarial learning for joint distribution matching. In *Advances in neural information processing systems* (pp. 5501–5509).

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Liu, Y., Gupta, A., Abbeel, P., & Levine, S. (2018). Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 ieee international conference on robotics and automation (icra)* (pp. 1118–1125). IEEE.

Mahelona, K., Glickman, J., Epstein, A., Brock, Z., Siripong, M., & Moyer, K. (2007). Darpa grand challenge.

Martinez-Gonzalez, P., Oprea, S., Garcia-Garcia, A., Jover-Alvarez, A., Orts-Escolano, S., & Garcia-Rodriguez, J. (2018). Unrealrox: An extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation. *arXiv preprint arXiv:1810.06936*.

McCormac, J., Handa, A., Leutenegger, S., & Davison, A. J. (2017). Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *Cvpr*.

Metta, G., Sandini, G., Vernon, D., Natale, L., & Nori, F. (2008). The iCub humanoid robot: An open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems* (pp. 50–56). ACM.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., & Cernockỳ, J. (2011). Empirical evaluation and combination of advanced language modeling techniques. In *Interspeech*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Nips*.

Mirza, M. & Osindero, S. (2014). Conditional generative adversarial nets. *Arxiv*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ..., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Mordatch, I., Mishra, N., Eppner, C., & Abbeel, P. (2016). Combining model-based policy search with online model learning for control of physical humanoids. In *Robotics and automation (icra), 2016 ieee international conference on* (pp. 242–248). IEEE.

Nagabandi, A., Kahn, G., Fearing, R. S., & Levine, S. (2017). Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*.

Nakazawa, K., Quirk, M. C., Chitwood, R. A., Watanabe, M., Yeckel, M. F., Sun, L. D., ..., Wilson, M. A., et al. (2002). Requirement for hippocampal ca3 nmda receptors in associative memory recall. *Science*, *297*(5579), 211–218.

Nister, D. & Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer vision and pattern recognition, 2006 ieee computer society conference on* (Vol. 2, pp. 2161–2168). Ieee.

Niu, C., Li, J., & Xu, K. (2018). Im2struct: Recovering 3d shape structure from a single RGB image. *CVPR*.

Oh, J., Singh, S., Lee, H., & Kohli, P. (2017). Zero-shot task generalization with multi-task deep reinforcement learning. In *Icml*.

OpenAI Universe. (2016). https://universe.openai.com/.

Pasquale, G., Ciliberto, C., Odone, F., Rosasco, L., Natale, L., & dei Sistemi, I. (2015). Teaching iCub to recognize objects using deep convolutional neural networks. *Proc. Work. Mach. Learning Interactive Syst*, 21–25.

Paull, L., Tani, J., Ahn, H., Alonso-Mora, J., Carlone, L., Cap, M., ..., Fang, Y., et al. (2017). Duckietown: an open, inexpensive and flexible platform for autonomy education and research. In *Robotics and automation (icra), 2017 ieee international conference on* (pp. 1497–1504). IEEE.

Pecka, M., Zimmermann, K., Petrlík, M., & Svoboda, T. (2018). Data-driven policy transfer with imprecise perception simulation. *arXiv preprint arXiv:1804.01953*.

Peng, X. B., Andrychowicz, M., Zaremba, W., & Abbeel, P. (2017). Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint arXiv:1710.06537*.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., & Courville, A. (2017). Film: Visual reasoning with a general conditioning layer. *arXiv preprint arXiv:1709.07871*.

Pfeifer, R. & Scheier, C. (2001). *Understanding intelligence*. MIT press.

Pfister, H., Zwicker, M., van Baar, J., & Gross, M. (2000). Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on computer graphics and interactive techniques*. SIGGRAPH '00.

Pickem, D., Glotfelter, P., Wang, L., Mote, M., Ames, A. D., Feron, E., & Egerstedt, M. (2017). The robotarium: A remotely accessible swarm robotics research testbed.

Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., & Torralba, A. (2018). Virtualhome: Simulating household activities via programs. In *Cvpr*.

Punjani, A. & Abbeel, P. (2015). Deep learning helicopter dynamics models. In *Robotics and automation (icra), 2015 ieee international conference on* (pp. 3223–3230). IEEE.

Qi, C. R., Su, H., Mo, K., & Guibas., L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Computer Vision and Pattern Recognition (CVPR)*.

Quine, W. V. O., Churchland, P. S., & Føllesdal, D. (2013). *Word and object*. MIT press.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations (ICLR)*.

Rajeswar, S., Mannan, F., Vazquez, D., Golemo, F., Nowrouzezahrai, D., & Courville, A. (2019). Pix2scene: Learning implicit 3d representations from images. *7th International Conference on Learning Representations (ICLR) 2019*.

Rasmussen, C. (2006). *Gaussian processes for machine learning*. Cambridge, Mass: MIT Press.

Rezende, D. J., Eslami, S. M. A., Mohamed, S., Battaglia, P., Jaderberg, M., & Heess, N. (2016). Unsupervised Learning of 3D Structure from Images. *Advances in Neural Information Processing Systems (NIPS)*.

Rolf, M., Steil, J. J., & Gienger, M. (2010). Goal babbling permits direct learning of inverse kinematics. *IEEE Transactions on Autonomous Mental Development*, *2*(3), 216–229.

Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to sift or surf. In *Computer vision (iccv), 2011 ieee international conference on* (pp. 2564–2571). IEEE.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ..., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, *115*(3), 211–252.

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., & Hadsell, R. (2016). Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*.

Savva, M., Chang, A. X., Dosovitskiy, A., Funkhouser, T., & Koltun, V. (2017). MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv:1712.03931*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd international conference on machine learning (icml-15)* (pp. 1889–1897).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., & Webb, R. (2017). Learning from simulated and unsupervised images through adversarial training. In *The ieee conference on computer vision and pattern recognition (cvpr)* (Vol. 3, p. 6).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ..., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., ..., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Sinapov, J., Schenck, C., & Stoytchev, A. (2014). Learning relational object categories using behavioral exploration and multimodal perception. In *Ieee int. conf. on robotics and automation* (pp. 5691–5698).

Singh, A., Sha, J., Narayan, K. S., Achim, T., & Abbeel, P. (2014). Bigbird: A large-scale 3d database of object instances. In *Ieee int. conf. on robotics and automation* (pp. 509–516).

Smith, L. & Yu, C. (2008). Infants rapidly learn word-referent mappings via cross-situational statistics. *Cognition*, *106*(3), 1558–1568.

Soltani, A. A., Huang, H., Wu, J., Kulkarni, T. D., & Tenenbaum, J. B. (2017). Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. *Computer Vision and Pattern Recognition (CVPR)*.

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., & Funkhouser, T. (2017). Semantic scene completion from a single depth image. *CVPR*.

Stein, G. J. & Roy, N. (2018). Genesis-rt: Generating synthetic images for training secondary real-world tasks. In *2018 ieee international conference on robotics and automation (icra)* (pp. 7151–7158). IEEE.

Sung, J., Ponce, C., Selman, B., & Saxena, A. (2011). Human activity detection from RGBD images. *plan, activity, and intent recognition*, *64*.

Taha, A. A. & Hanbury, A. (2015). An efficient algorithm for calculating the exact hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Taigman, Y., Polyak, A., & Wolf, L. (2016). Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200*.

Tani, J., Paull, L., Zuber, M. T., Rus, D., How, J., Leonard, J., & Censi, A. (2016). Duckietown: an innovative way to teach autonomy. In *International conference edurobotics 2016* (pp. 104–121). Springer.

Tobin, J. [Josh], Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *arXiv preprint arXiv:1703.06907*.

Tobin, J. [Joshua], Zaremba, W., & Abbeel, P. (2017). Domain randomization and generative models for robotic grasping. *arXiv preprint arXiv:1710.06425*.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent robots and systems (iros), 2012 ieee/rsj international conference on* (pp. 5026–5033). IEEE.

Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., & Darrell, T. (2014). Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*.

Vatakis, A. & Pastra, K. (2016). A multimodal dataset of spontaneous speech and movement production on object affordances. *Scientific Data*. Retrieved from http://dx.doi.org/10.1038/sdata.2015.78

Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2017). Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *TPAMI*, *39*(4), 652–663.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3), 229–256.

Woodcock, R., Mather, N., & McGrew, K. (2001). Woodcock johnson iii - tests of cognitive skills. *Riverside Pub*.

Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, W., & Tenenbaum, J. s. (2016). Marrnet: 3d shape reconstruction via 2.5d sketches. *ArXiv*.

Wu, J., Xue, T., Lim, J., Tian, Y., Tenenbaum, J., Torralba, A., & Freeman, W. s. (2016). Single image 3d interpreter network. *ArXiv*.

Wu, J., Zhang, C., Xue, T., Freeman, W. T., & Tenenbaum, J. B. (2016). Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. *Advances in Neural Information Processing Systems (NIPS)*.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. *Computer Vision and Pattern Recognition (CVPR)*.

Xia, F., R. Zamir, A., He, Z., Sax, A., Malik, J., & Savarese, S. (2018). Gibson Env: Real-world perception for embodied agents. In *Computer vision and pattern recognition (cvpr), 2018 ieee conference on*. IEEE.

Yan, C., Misra, D., Bennnett, A., Walsman, A., Bisk, Y., & Artzi, Y. (2018). Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357*.

Yang, L., Liang, X., Wang, T., & Xing, E. (2018). Real-to-virtual domain unification for end-to-end autonomous driving. In *Proceedings of the european conference on computer vision (eccv)* (pp. 530–545).

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems* (pp. 3320–3328).

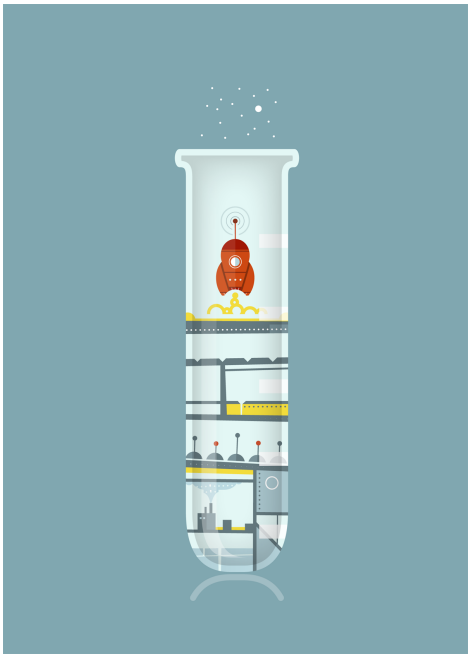
Zagal, J. C. & Ruiz-Del-Solar, J. (2007). Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*, *50*(1), 19–39.

Zagal, J. C., Ruiz-del-Solar, J., & Vallejos, P. (2004). Back to reality: Crossing the reality gap in evolutionary robotics. *IFAC Proceedings Volumes*, *37*(8), 834–839.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., & Farhadi, A. (2017). Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. In *ICRA*.

Zilly, J., Boyarski, A., Carvalho, M., Abarghouei, A. A., Amplianitis, K., Krasnov, A., ..., Pérez, C. S., et al. (2017). Inspiring Computer Vision System Solutions. *arXiv preprint arXiv:1707.07210*.



This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Egenolff-Berner Garamond, a revival of Claude Garamont's humanist typeface. The above illustration, *Science Experiment 02*, was created by Ben Schlitter and released under CC BY-NC-ND 3.0. A template that can be used to format a PhD dissertation with this look & feel has been released under the permissive AGPL license, and can be found online at github.com/suchow/Dissertate or from its lead author, Jordan Suchow, at suchow@post.harvard.edu.