Imperial College London

Department of Computing

Tightly-coupled Manipulation Pipelines: Combining Traditional Pipelines and End-to-End Learning

Stephen Lloyd James

4th May 2021

Supervised by Prof. Andrew Davison

Abstract

Traditionally, robot manipulation tasks are solved by engineering solutions in a modular fashion — typically consisting of object detection, pose estimation, grasp planning, motion planning, and finally run a control algorithm to execute the planned motion. This traditional approach to robot manipulation separates the hard problem of manipulation into several self-contained stages, which can be developed independently, and gives interpretable outputs at each stage of the pipeline. However, this approach comes with a plethora of issues, most notably, their generalisability to a broad range of tasks; it is common that as tasks get more difficult, the systems become increasingly complex.

To combat the flaws of these systems, recent trends have seen robots visually learning to predict actions and grasp locations directly from sensor input in an end-to-end manner using deep neural networks, without the need to explicitly model the in-between modules. This thesis investigates a sample of methods, which fall somewhere on a spectrum from pipelined to fully end-to-end, which we believe to be more advantageous for developing a general manipulation system; one that could eventually be used in highly dynamic and unpredictable household environments.

The investigation starts at the far end of the spectrum, where we explore learning an end-to-end controller in simulation and then transferring to the real world by employing domain randomisation, and finish on the other end, with a new pipeline, where the individual modules bear little resemblance to the 'traditional' ones. The thesis concludes with a proposition of a new paradigm: **Tightly-coupled Manipulation Pipelines (TMP)**. Rather than learning all modules implicitly in one large, end-to-end network or conversely, having individual, pre-defined modules that are developed independently, TMPs suggest taking the best of both world by tightly coupling actions to observations, whilst still maintaining structure via an undefined number of learned modules, which do not have to bear any resemblance to the modules seen in 'traditional' systems.

Acknowledgements

I am very thankful for the support given to me by many people over the course of my studies, without which this work would not have been possible.

I cannot thank my supervisor, Prof. Andrew Davison, enough. He has taught me a great deal on this PhD journey. In particular, he has put trust in me to pursue areas of research that fell outside the usual scope of the lab. I would also like to say a special thank you to Dr. Edward Johns for supervising my undergraduate project that lead me on the path to pursue my PhD in robot manipulation.

I am thankful for the advice and discussions with all other members of the Dyson Robotics Lab, past and present: Tristan Laidlow, Daniel Lenton, Kentaro Wada, Michael Bloesch, Dorian Hennings, Charlie Houseago, Zoe Landgraf, Robert Lukierski, John McCormac, Sajad Saeedi, Edgar Sucar, Shuaifeng Zhi, Patrick Bardow, Ankur Handa.

Iosifina Pournara was particularly helpful to me, organising travel to conferences and ensuring that I had access to the equipment I needed, as well as providing general support, advice and encouragement.

Finally, I would like to thank my wife for her unconditional love and unyielding support during the PhD. Thank you for supporting me along this long journey, and for making the sacrifices that got me to where I am now.

I am very appreciative of Dyson Technology Ltd. for funding this research.

Contents

CHAPTER 1

Introduction

Contents

Robot manipulation is hard. In order to truly understand the difficulty of a concept that comes so naturally to humans, one must fully *understand* what it means to manipulate.

Consider the task of hammering a nail into a block of wood. The task can roughly be broken up into the following steps: locate the hammer, grasp the hammer, locate the nail, grasp the nail, locate the wood, hold the nail onto the wood, and finally use the hammer by swinging it at the nail with a large force. The immediate thing to notice is that this one task actually consists of many, smaller tasks, that even by themselves are very challenging for a robot.

A common theme in this task is to locate an object, potentially in a cluttered environment. This requires a thorough 3D understanding of the surrounding area; an understanding that extends to new and unfamiliar environments. Even in this first stage, the robot must be able to deal with several potential

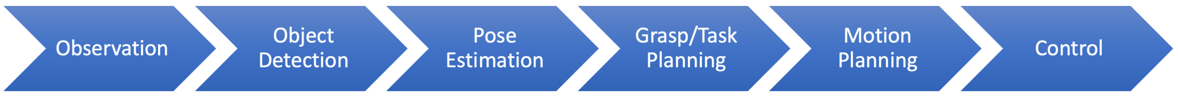


Figure 1.1: An example of a typical manipulation pipeline for an unstructured task.

modes of failure: What if the hammer is slightly occluded? What if there are two hammers? What if there is no hammer? What if the hammer was a toy hammer? It is clear then, that 3D understanding is, by itself, a very hard problem.

Another common theme from the hammering task is the ability to grasp objects. A single object can be grasped in several ways, although in practice depending on the object, only a particular subset of these grasps are performed. For example, knives are grasped using the handle, and grasping the blade itself would lead to harm. It is therefore vital for both humans and robots alike to be aware of what the object is before attempting a grasp in order to allow the correct manipulation task to be performed. The hammering task illustrates the grasping problem well, because it involves grasping two very different objects: the hammer, which must be grasped in an appropriate way such that it can be used later in the task, and the nail, which in comparison to the hammer is very small, and requires dexterous grasping.

With hammer and nail in hand, the hammering process can commence. During the process, the robot needs to keep track of both hammer and nail in order to correctly bring the hammer onto the nail, and monitor the surrounding environment for potential hazards. It is clear that accuracy at this stage of the task is fundamental, as a small mistake due to a lack of control could lead to task failure and potentially cause damage to the robot, onlookers, or the environment. Performing actions with skill is also important for recovering from unanticipated situations, such as a nail bending during the hammering process. All of these challenges sub-problems define the greater robot manipulation problem.

As has been the case for decades, many robots predominately exist in the industrial domain, and do not have to concern themselves with the complexities discussed above. These industrial robots have the ability to lift massive loads, move with incredible speed, and perform complex sequences of actions with pin-point accuracy; capabilities made possible due to expensive motors and sensors, finely-tuned controllers, and strong materials. Despite their successful application in this industrial domain, they can give a skewed representation of what is currently possible in an everyday human environment. These robots can only operate successfully in a pre-defined, repetitive manner and cannot operate outside of their constrained environment. In reality, the world is highly dynamic and unpredictable, and all of these robots would fail to adapt when they are presented with new and unfamiliar environments. How then, do we get manipulating robots into our homes?

Until very recently, the predominant way of building adaptive manipulation systems was in a modular, pipelined fashion, that includes object detection and recognition, 6D pose estimation, grasp planning, motion planning, and control. Figure 1.1 shows a simplified example of this pipeline, but in practice these pipelines can contain many other modules. The use of a pipelined method like this comes with a number of benefits. The hard problem of manipulation is broken into several self-contained stages, which can be developed independently. Because of this modularity, each stage gives an interpretable output which can be tested and understood in isolation. However, this modularity can also come at a high cost. For starters, if there is a mistake made in the earlier part of the pipeline, then the effect can trickle through to the rest of the modules. An example of this would be a mistake in the pose estimation phase, which would then mean that the scene representation is incorrect, leading to an incorrect motion plan and potentially a collision during control. While having an interpretable input and output can be beneficial, it can also be a hindrance; a conscious decision needs to be made beforehand about what the input-output representation for each of the modules should be and how they should be stored. This opening up the possibility of choosing a sub-optimal representation that may be detrimental to task performance. A possibly greater issue is the generalisability of a pipeline like this. It is usually unlikely that one system can be applied to many different types of tasks. For example in the case of moving from a pick-and-place task to a sweeping task, the scene understanding, planning and control modules would need to be redesigned. Finally, as tasks get more difficult, the pipeline becomes increasingly complex.

In light of this, an emerging alternative is an end-to-end paradigm that bypasses the intermediate steps and maps observations directly to robot actions through a deep neural network. The intuition here being that the 'traditional' modules are embedded in the weights of the network, without the need to explicitly model them. Given the success of going from modular to end-to-end in to other fields, such as computer vision and natural language processing, this may initially seem like a good idea. However, given the complexity of the functions we are trying to model, these end-to-end methods tend to be incredibly data hungry. Not only that, but the data needed for robot manipulation is arguable much more difficult to collect than conventional computer vision problems. One reason for this is that data collection usually has to be done on the platform itself, which can be expensive in both time and hardware cost. One way to circumvent this can be to leverage the power of simulation to produce large amounts of data. Here we have access to a large amount of data in a safe environment without the need for human supervision. We can train in this environment, and then deploy in the real world. However, due to both the visual and dynamic discrepancies between the simulated world and the real world, direct transfer is not possible, motivating the need for sim-to-real methods.

Naturally, there are trade-offs when choosing between pipelined and end-to-end methods for robot manipulation. This thesis investigates a range of methods which fall somewhere on a spectrum from pipelined to fully end-to-end. Our investigation focuses on areas of the spectrum that we believe to be more advantageous for developing a general manipulation system; one that could eventually be used in highly dynamic and unpredictable household environments. Our areas of focus tend towards the end-to-end side of the spectrum, as is shown in Figure 1.2. As is evident by the hammering task outlined previously, the area of robot manipulation is broad; too broad to be tackled in a single thesis. We therefore limit our scope of manipulation to consist only of a single-arm setup that is affixed to a stationary surface. The tasks we consider mostly consist of rigid objects, and we disregard tasks that usually contain complex physics (excluding Chapter 4). The investigation starts at the far end of the spectrum, where we explore learning an end-to-end controller in simulation and then transferring to the real world by employing domain randomisation [James et al., 2017a]. We train this end-to-end (image to velocity) imitation learning method on a multi-stage task, which is analogous to a simple tidying routine, without having seen a single real image. This involves locating, reaching for, and grasping a cube, then locating a basket and dropping the cube inside. To achieve this, robot trajectories are computed in a simulator, to collect a series of control velocities which accomplish the task. Then, a convolutional neural network (CNN) is trained to map observed images to velocities, using domain randomisation to enable generalisation to real world images. Results show that we are able to successfully accomplish the task in the real world with the ability to generalise to novel environments, including those with dynamic lighting conditions, distractor objects, and moving objects, including the basket itself. The result is a highly scalable system that is capable of learning long-horizon tasks that had not been shown with state-of-the-art in end-to-end robot control.

The end-to-end method mentioned above fared well when dealing with rigid objects (such as the cube), but how would it fare when faced with deformable objects? Due to the large configuration space of deformable objects, solutions using traditional modelling approaches require significant engineering work. Perhaps then, bypassing the need for explicit modelling and instead learning the control in an end-to-end manner serves as a better approach? Until now, no work had explored whether it is possible to learn and transfer deformable object control policies from simulation to reality. We believe that if sim-to-real methods are to be employed further, then it should be possible to learn to interact with a wide variety of objects, and not only rigid objects. To that end, in subsequent work, we use a combination of state-of-the-art deep reinforcement learning algorithms to solve the problem of manipulating deformable objects (specifically cloth) [Matas et al., 2018]. We evaluate our approach on three tasks — folding a towel up to a mark, folding a face towel diagonally, and draping a piece of cloth over a hanger. We show that we are able to fully train in simulation with domain randomisation, and then successfully deployed in the real world without having seen any real deformable objects.

Although the two approaches highlighted above have been successful, they have only focused on learning a single task, from scratch, with no way of leveraging the knowledge to learn other tasks more efficiently. Much like humans, robots should have the ability to leverage knowledge from previously learned tasks in order to learn new tasks quickly in new and unfamiliar environments. With this in mind, we introduce Task-Embedded Control Networks (TECNets) [James et al., 2018], which employ ideas from metric learning in order to create a task embedding that can be used by a robot to learn new tasks from one or more demonstrations. We show that our approach can also be used in conjunction with domain randomisation to train our few-shot learning ability in simulation and then deploy in the real world without any additional training. Once deployed, the robot can learn new tasks from a single real-world teleoperated demonstration.

The above work gives an intuitive way of providing demonstrations to a robot. However, humans can naturally learn to execute a new task by seeing it performed by other individuals once, and then reproduce it in a variety of configurations. Endowing robots with this ability of imitating humans from third person is a very immediate and natural way of teaching new tasks. We extend TECNets further to infer control polices by observing videos of humans performing the desired task [Bonardi et al., 2020]. Importantly, we do not use a real human arm to supply demonstrations during training, but instead leverage domain randomisation in an application that has not been seen before: sim-to-real transfer on humans. Upon evaluating our approach on pushing and placing tasks in both simulation and in the real world, we show that in comparison to a system that was trained on real-world data we are able to achieve similar results by utilising only simulation data.

Despite achieving high success on one-shot imitation learning with TECNets, variation across tasks was limited. This was predominantly down to the fact that we were in uncharted territory: there did not exist a suitable benchmark or standard set of tasks for the few-shot manipulation community. It was this that motivated the creation of a challenging new benchmark and learning-environment for robot learning: RLBench [James et al., 2020]. The benchmark features 100 completely unique, hand-designed tasks, ranging in difficulty from simple target reaching and door opening to longer multi-stage tasks, such as opening an oven and placing a tray in it. We provide an array of both proprioceptive observations and visual observations, which include rgb, depth, and segmentation masks from an over-the-shoulder stereo camera and an eye-in-hand monocular camera. Uniquely, each task comes with an infinite supply of demos through the use of motion planners operating on a series of waypoints given during task creation time; enabling an exciting flurry of demonstration-based learning possibilities. RLBench has been designed with scalability in mind; new tasks, along with their motion-planned demos, can be easily created and then verified by a series of tools, allowing users to submit their own tasks to the RLBench task repository. This large-scale benchmark aims to accelerate progress in a number of vision-guided manipulation research areas, including: reinforcement learning, imitation learning, multi-task learning, geometric computer vision, and in particular, few-shot learning.

Upon evaluating many image-based state-of-the-art reinforcement learning algorithms on RLBench, it was clear these approaches fail catastrophically in these complex, sparsely-rewarded tasks. This motivated the need for a more structured approach to the problem. With this in mind, we present our Attention-driven Robotic Manipulation (ARM) algorithm [James and Davison, 2021], which is a general manipulation algorithm that can be applied to a range of real-world sparse-rewarded tasks without any prior task knowledge. ARM splits the complex task of manipulation into a 3 stage pipeline: (1) a Q-attention agent extracts interesting pixel locations from RGB and point cloud inputs, (2) a next-best pose agent that accepts crops from the Q-attention agent and outputs poses, and (3) a control agent that takes the goal pose and outputs joint actions.

There is a clear evolution through the thesis, where in the initial chapters, we scrap the 'traditional' manipulation pipeline, and start afresh with an end-to-end approach. As the chapters progress, we gradually modularise, and finally end up with a new pipeline, where the individual modules bear little resemblance to the 'traditional' ones. With that in mind, the thesis culminates with the proposition of a new paradigm: **Tightly-coupled Manipulation Pipelines (TMP)**. Rather than learning all modules implicitly in one large, end-to-end network or conversely, having individual, pre-defined modules that are developed independently, TMPs suggest taking the best of both world by tightly coupling actions to observations, whilst still maintaining structure via an undefined number of learned modules, which do not have to bear any resemblance to the modules seen in 'traditional' systems. This tight coupling of TMPs can be achieved either implicitly through gradients flowing through each of the modules, such as in Chapter 5, or implicitly via a shared reward, such as in Chapter 7. Figure 1.2 illustrates where on the spectrum this TMP paradigm falls, and where the individual methods presented in this thesis fall. Note that due to the limited time of the PhD, our exploration focuses on methods that are to the left of the spectrum, focusing mostly on end-to-end rather than pipelined; we reflect on this in Chapter 8.

A brief historical review of manipulation is given below. References to more thorough historical surveys will be given for further reading, as only a general outline of key milestones is given along with more specific discussions on work closely related to the research presented in this thesis.

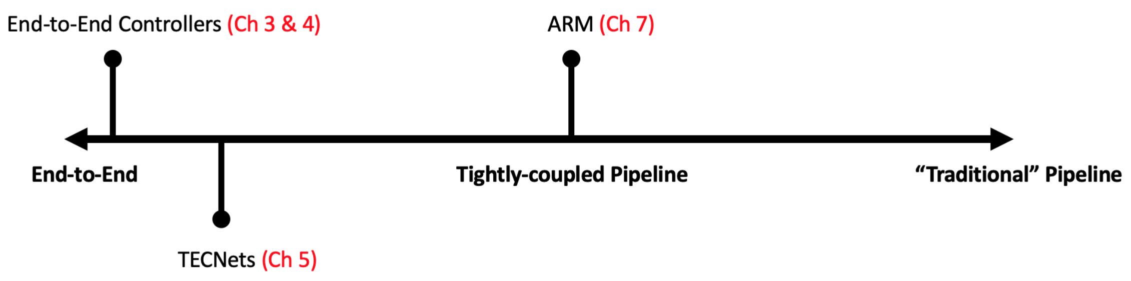


Figure 1.2: The spectrum from fully end-to-end to pipelined. The branches indicate where each piece of work presented in the thesis lies on the spectrum.

1.1 Manipulation Through The Years

The origin of robotic manipulation research can be traced back as far as the 1950s, where teleoperated manipulators were developed for handling nuclear materials [Goertz, 1952b, Goertz, 1952a, Goertz, 1954, Goertz and Thompson, 1954, Goertz, 1963]. Fast forward 70 years, and teleportation is still an active area of research in a number of fields, including surgical systems [Choi et al., 2018], bomb disposal [Lisle, 2020], underwater vehicles [Havoutis and Calinon, 2019], and space manipulators [Yoon et al., 2004].

The emergence of industrial robotics began in the 1960s with the founding of Unimation and the first industrial robot, the Unimate [Devol, 1961]. Today, the global industrial robot market is estimated to be worth in excess of $21 Billion [Insights, 2020]. The first examples of what we might call general robots today, came from Stanford Research Institutes (SRI) seminal autonomous mobile agent, Shakey [Fikes et al., 1972, Nilsson, 1984], which applied emerging automated theorem proving techniques in robot planning. Although Shakey did not perform manipulation, its ability to perceive, plan, and then act, laid the foundation of the manipulation pipelines that we discussed in the opening section. Naturally these pipelines have become more sophisticated over the past 50 years, incorporating additional sensory inputs, developing rich scene understanding, more complex planning, a greater dexterous control. Research in pipelined manipulation usually focuses on presenting improvements and evaluation of modules in isolation (e.g. presenting a novel pose estimation system), before plugging them into a standard manipulation pipeline for a final demo. In the following sections, we will briefly overview progress in these areas. Note that there are a large number of other areas of manipulation which we consider out of scope of this thesis, including: in-hand manipulation, non-prehensile manipulation, and dual-arm manipulation, to name but a few.

1.1.1 Scene Understanding

Understanding *what* objects are in the scene, and *where* they are, is a vital step in not only robot manipulation tasks [Zhu et al., 2014, Collet et al., 2011], but also many other areas, including medical imaging [Ralovich et al., 2014] and augmented reality [Hagbi et al., 2011]. *Object detection*, or *recognition*, is the process of finding objects within an image, whilst *6D pose estimation* determines the transformation of that object. Both are well studied problems in the computer vision community, with a vast number of solutions. This section will give a brief overview of four methods: *template-based methods*, *sparse feature-based methods*, *dense methods*, and more recently *end-to-end methods*.

Template-based methods are one of the earliest approaches to object detection and pose estimation. Traditionally, these methods involve generating templates by collecting images of the object from varying viewpoints in an offline training stage and then scanning the template across an image to find the best match using a distance measure [Huttenlocher et al., 1993, Steger, 2001]. These methods are sensitive to clutter, occlusions, and lighting conditions, leading to a number of false positives, which in turn requires greater post processing. LINEMOD [Hinterstoisser et al., 2011, Hinterstoisser et al., 2012a] attempts to improve detection of texture-less objects in cluttered scenes by generating templates that combine silhouette gradient orientations from RGB images and surface normal orientations from depth images. Extensions of this method involve the use of 3D models to generate many templates of the objects from different viewing angles [Hinterstoisser et al., 2012b], as well as an effort to increase their speed by using a cascade framework [Rios-Cabrera and Tuytelaars, 2013].

Sparse feature-based methods have been a popular alternative to template-based methods for a number of years [Lowe, 2001, Nister and Stewenius, 2006, Philbin et al., 2007]. These methods are concerned with extracting scale invariant points of interest from images, describing them with local descriptors (such as SIFT [Lowe, 2004] or SURF [Bay et al., 2008]), and then storing them in a database to be later matched with. At test time, these descriptors are used with methods such as RANSAC [Fischler and Bolles, 1981] to gain pose estimates. This processing pipeline can be seen in systems such as MOPED [Collet et al., 2011]; in this system, structure from motion is used to merge the information from each training image into a sparse 3D model, which is used to obtain the pose based on 3D point-to-point correspondences between a stored model and the test model. MOPED has inspired further work, such as the addition of depth information at multiple stages in the pipeline [Tang et al., 2012]. Recently, instead of using predefined hand-crafted features, features have instead been learned through examples [Holzer et al., 2012, Rosten et al., 2010]. As these methods look for features in the input data, they make the assumption that objects have sufficient texture, leading to poor performance on texture-poor objects or failure on texture-less objects.

With the increase in affordable RGB-D cameras, dense methods have become increasingly popular for object and pose recognition [Drost et al., 2010, Shotton et al., 2013, Brachmann et al., 2014]. These methods involve construction of a 3D point-cloud of a target object, and then matching this with a stored model using popular algorithms such as Iterative Closest Point (ICP) [Besl and McKay, 1992].

Object detection in-itself has a large deep learning community around it. The advancements in object recognition are in fact due to rapid progress in the image classification problem. In 1994, after 8 years of iterative work, Yann LeCun published one of the very first convolutional neural networks (CNNs): **LeNet5** [LeCun et al., 1998]. It was not until 2012 that the deep learning boom ignited with the release of **AlexNet** [Krizhevsky et al., 2012] — a much deeper and wider version of LeNet5. What followed was a series of publications that sought to outperform previous networks on datasets such as MNIST [LeCun et al., 2010] and ImageNet [Russakovsky et al., 2015b]. Such networks included **ZF Net** [Zeiler and Fergus, 2014], which achieved a lower error rate whilst using fewer training examples than AlexNet. Moreover, the authors reasoned that using the larger 11×11 filter size in the first layer skips large amounts of relevant information, and therefore using smaller filters with a smaller stride would retain that information (7×7 filter size in this case). In 2014, **VGG Net** [Simonyan and Zisserman, 2014] took this further by using only 3×3 filters in their network, reasoning that the combination of two 3×3 filters have an effective receptive field as a 5×5 filter, and likewise three 3×3 filters have an effective receptive field as a 7×7 filter. This results in keeping the benefits of larger filters while reducing the number of parameters.

Instead of stacking convolution layers on top of each other in order to improve performance, **GoogLeNet** [Szegedy et al., 2015] introduced the idea of an *inception module*. The inception module allows the following operations to be performed in parallel: a 1×1 convolution, allowing fine grained detail to be extracted, a medium and large sized convolution, which allows higher level features to be extracted, and a pooling operation to reduce spatial sizes and overfitting.

To-date, the deepest networks have been inspired by **ResNet** [He et al., 2016b], which spanned an incredible 152 layers and achieved record breaking results, not only in classification, but also in detection and localization. The key contribution, other than the huge number of layers, was the *residual block*. In a traditional convolutional layer, an input x is transformed by some function $f(x)$. In a residual block, the output from $f(x)$ is instead treated as a *delta* that is added to the input x to give a slightly altered representation $g(x) = f(x) + x$.

Alongside the advances in image classification, object recognition was impacted heavily in 2013 by the release of *Regions with CNN features* (R-CNN) [Girshick et al., 2014]. *Selective search* [Uijlings et al., 2013] was used on input images to extract 2000 region proposals which are then fed into a CNN to produce image features that are used as input to a class-specific linear SVM to output a classification.

From the success of R-CNN followed two extensions: Fast R-CNN [Girshick, 2015] and Faster R-CNN [Ren et al., 2015]. The original R-CNN was computationally expensive and required 53 seconds per image. Fast R-CNN [Girshick, 2015] improved on this by reducing training time by combining the classification and localisation loss into a single multi-task loss, and by using a *region of interest* (RoI) pooling layer meaning that images do not need to be scaled before entering the CNN for feature extraction. The next version, Faster R-CNN [Ren et al., 2015], simplified the complex training pipelines of its predecessors by producing region proposals through a *region proposal network* (RPN) that was introduced after the convolutional layers, created a single, unified network for object detection.

From the success of object detection came Mask R-CNN [He et al., 2017] for instance segmentation. The model works by first performing object detection to draw bounding boxes around each instance of a class, and then performs semantic segmentation on each of the bounding boxes. Mask R-CNN is now widely used in many robotic applications [Wang et al., 2019, Wada et al., 2020].

1.1.2 Grasp Planning

Robotic grasping is a well studied problem [Bohg et al., 2014], and is concerned with finding a grasp point on an object that results in the ability to pick up the object. The method usually depends on the type of gripper that is used.

Parallel-jaw grippers are very popular. Traditionally, grasping was usually solved analytically, where 3D meshes of objects would be used to compute the stability of a grasp against external wrenches [Prattichizzo and Trinkle, 2008, Rodriguez et al., 2012] or constrain the object's motion [Rodriguez et al., 2012]. These solutions often assume that the same, or similar objects will be seen during testing, such that point clouds of the test objects can be matched with stored objects based on visual and geometric similarity [Brook et al., 2011, Ciocarlie et al., 2014, Hernandez et al., 2016, Hinterstoisser et al., 2011, Kehoe et al., 2013]. Due to this limitation, data-driven methods have become the dominant way to solve grasping [Lenz et al., 2015, Mahler et al., 2017]. These methods commonly make use of either hand-labeled grasp positions [Lenz et al., 2015, Kappler et al., 2015], self-supervision [Pinto and Gupta, 2016a], or predicting grasp outcomes [Levine et al., 2016b]. To train these solutions, data is needed, which can come from datasets such as the Cornell grasping dataset [Jiang et al., 2011]; this dataset features RGB and depth images of objects where suitable grasp points has been manually-labelled. The dataset has been used to train a number of solutions which regress to a probability that a particular pose will be graspable [Lenz et al., 2015, Redmon and Angelova, 2015]. Acquiring large amount of manually-labelled data can be time consuming, and so a number of solutions exist to combat this. Automatic dataset generation through random grasps is an attractive approach. After 700 hours of running, a dataset of 50,000 had been collected through performing random grasps within regions of interest [Pinto and Gupta, 2016b]. Further work collected 800,000 grasps using 14 robots that were run for two months [Levine et al., 2016c]. Simulators are another option when it comes to data generation. The *GraspIt!* simulator [Miller and Allen, 2004] is a popular choice for evaluating grasps [Varley et al., 2015], but most work is restricted to the virtual domain.

Suction grippers are an attractive alternative to parallel-jaw grippers, and have been incredibly popular in the Amazon Picking Challenge (APC) due to their simplicity and effectiveness in picking up a wide variety of objects [Eppner et al., 2015, Hernandez et al., 2016]. For suction grippers, a popular method involves computing surface normals in a grid-like pattern across the surface of a smoothed point cloud segment in order to generate grasp candidates, which are then ranked according to some constraints [Leitner et al., 2016, Hernandez et al., 2016].

1.1.3 Motion Planning

Motion planning is concerned with going from one pose to another, whilst avoiding obstacles and taking into account constraints, such as joint limits, maximum velocity, and jerk. The origin of motion planning can be traced back to two major milestones. The first was by Lozano-Perez and Wesley, who developed the use of configuration space for collision-free motion planning [Lozano-Pérez and Wesley, 1979]. For a robot arm with n degrees of freedom (DoF), a configuration space (C-space) is defined as a coordinate system with one dimension per DoF. The free space is defined to be the subset of the C-space comprising all configurations that do not place the arm in an obstacle. The path planning problem is then to find a curve in the free space that connects the start configuration with the goal configuration. The other milestone was that of sampling-based planners [Kavraki et al., 1996]. Rather than calculating all free configurations and planning in free space as is done in C-space planning, sampling-based planners instead continuously sample in the configuration space whilst checking for collisions as a path is generated. There are now many sampling-based planners in use today, including PRM [Kavraki et al., 1996], EST [Hsu et al., 1997], RRT [Kuffner and LaValle, 2000], RRT* [Karaman and Frazzoli, 2011] and BFMT* [Starek et al., 2015], to name but a few. For a detailed survey of sampling-based planning, see [Elbanhawi and Simic, 2014].

1.1.4 Control

The final stage of the pipeline involves taking a series of high-level actions (e.g. joint positions, joint velocities, end-effector poses) and then apply techniques from control theory to convert these actions into low-level commands that drive the actuators. In robot manipulation, this is done in a closed-loop form, where inputs (e.g. voltage applied to an electric motor) effect the outputs of the system (e.g. the speed or torque of the motor) which are measured by sensors, that cause the controller to alter its behaviour in order to reach a desired speed or torque. From lowest level to highest level, a selection of action modes are: joint torques, joint velocities, joint positions, and end-effector poses. Low-level actions, such as torque control, give much finer, predictable control, whilst higher-level actions, such as joint velocities, removes the need for earlier components in the pipeline to reason about the low-level dynamics of the system (e.g. gravity compensation, motor uncertainty, etc).

There are many tasks that go beyond moving an object from one place to another, and instead involve complex contact dynamics. Imagine a robot tasked with drawing a circle with a pencil on a piece of paper. If the table is slightly closer than expected, or the pencil is slightly longer than expected, then the pencil lead will break; conversely, if the table is further, or the pencil is shorter than expected, then the pencil will not make contact with the paper. This motivates the need for *compliant motion*; a control method that deals with uncertainties, and was first discussed in area of teleoperation [Goertz, 1952b]. One of the most common approaches to accomplish compliant behaviour is through impedance control [Hogan, 1984], where the goal is to achieve a desired impedance of the robot at the end effector, felt by the environment.

1.1.5 Benchmarking

Although manipulation belongs under the general field of AI, it has not seen the same speed of progress in comparison to other related fields, such as machine learning. One of the reasons that robotics has not progressed as fast is because, unlike machine learning, comparison across systems is difficult. Challenges such as the Amazon Picking Challenge (APC) [Correll et al., 2016] and DARPA Robotics Challenge (DRC) [Pratt and Manzo, 2013] are effective ways of driving progress, especially in two distinct areas: picking and storing, and task completion in dangerous and degraded environments. Despite this, they have a number of drawbacks, including their lack of reproducibility, their nature of being held once a year, and their restriction in the number of teams.

In addition to the general computer vision datasets that focus on perception [Russakovsky et al., 2015b, Lin et al., 2014, Everingham et al., 2015], there exist a number of datasets that are more applicable to the robot setting; including image and point-cloud datasets [Singh et al., 2014], a 2D/3D database for detection in clutter [Tejani et al., 2014], and datasets consiting of RGB-D images for object segmentation [Lai et al., 2013] and pose estimation [Rennie et al., 2016]. In comparison to perception, grasping is harder to benchmark as it is difficult to classify a solution as correct or incorrect. Nowadays, manipulation capabilities can be compared in simulation through tools such as OpenGRASP [Ulbrich et al., 2011] and VisGraB [Popović et al., 2011], or through real-world experiments using datasets such as YCB [Calli et al., 2015] which can be tedious to compare.

The problem with benchmarking each of the components separately is that we do not know how the system would behave overall. That is, the perception benchmarks are disjoint from robot application, while the grasping benchmarks remove the emphasis on perception and planing. To combat this, attempts have been made to offer a benchmark that encompasses the entire picking task to allow comparison of complete robot systems [Leitner et al., 2016]. In Chapter 6 of this thesis, we present our own new benchmark called RLBench, which addresses the issues discussed above.

1.2 Rise of End-to-End Robot Manipulation

As has been mentioned at the beginning of this chapter, a recent trend has been to tightly couple sensing and action via a single deep learning network that maps observations directly to actions. This removes the need to implicitly model the scene and perform planning, and instead learn this implicitly in the weights of the network. This style of learning was made popular by the work of [Levine and Abbeel, 2014], where Guided Policy Search (GPS) [Levine and Koltun, 2013] was used to train deep visuomotor policies that mapped camera observations to motor torques of a PR2 robot. This method trained a range of manipulation tasks, including bottle-cap screwing, and inserting a block into a shape sorting cube in under 200 episodes; this is a large difference in comparison to other RL methods that require orders of magnitude more data [Lillicrap et al., 2015a, Schulman et al., 2015].

1.2.1 Guided Policy Search

Guided Policy Search (GPS) [Levine and Koltun, 2013, Levine and Abbeel, 2014], which in addition to classic control benchmarks, has achieved success in aerial flight [Zhang et al., 2016b, Kahn et al., 2016], and manipulation [Levine et al., 2016a, Montgomery and Levine, 2016, Montgomery et al., 2016]. GPS provides a means to optimise non linear policies, such as neural networks [Levine and Abbeel, 2014], without the need to compute policy gradients. GPS uses guiding samples produces by a 'teacher' algorithm, such as trajectory optimisers or trajectory-centric reinforcement learning methods, to train policies in a supervised manner. In addition to this, GPS adapts the guiding samples produced by the teacher so that they are more suited for training the final policy. Many methods have been proposed to improve the converge guarantees of the policy and teacher, including BADMM-GPS [Levine et al., 2016a], Mirror Descent GPS [Montgomery and Levine, 2016], as well as extentions that improve the teacher algorithm, such as adding ability to handle random initial states [Montgomery et al., 2016], and using model-free optimiser based on path integral stochastic optimal control (PI^2) [Chebotar et al., 2016].

Later work derived a new variant called Mirror Descent (MD-GPS) [Montgomery and Levine, 2016], which showed GPS could be interpreted as an approximate variant of mirror descent to simplify convergence guarantees in the convex and linear settings. BADMM-GPS and MD-GPS require a consistent set of initial states after each episode, which is not always a practical assumption outside of simulation, and limits the generalisation of the resulting global policy. Reset-Free GPS (RF-GPS) [Montgomery et al., 2016] is an extension to MD-GPS that allowed random initial starts, and as a result, the trained policies improved generalisability in comparison to prior methods, and with two orders of magnitude fewer samples.

So far we have seen the teacher algorithm implemented in a model-free method (such as the PI^2 method, which is efficient and fast), along with model-based methods (such as LQR with fitted local linear dynamics, which have the ability to handle complex dynamics). In order to combine the advantages of both these methods, a hybrid trajectory-centric RL method has been proposed to solve manipulation problems that prior methods were not able to solve, such as hitting a hockey puck into a goal and inserting a plug into a power socket [Chebotar et al., 2017]. Other interesting extensions to GPS include the addition of memory states that outperforms baselines that naively replaces the policy with a recurrent neural network [Zhang et al., 2016a], as well as adaptions to both BADMM-GPS and MD-GPS to work in a asynchronous manner distributed across 4 robots [Yahya et al., 2016].

As described above, guided policy search is a combination of trajectory optimisation or reinforcement learning with imitation learning; but these two fields have since had an abundance of end-to-end manipulation work in their own right, which we now discuss.

1.2.2 Deep Reinforcement Learning

Reinforcement learning (RL) [Sutton and Barto, 1998] has become a popular solution for learning the problem of control (see Section 2.5 for a brief overview of RL). It has been particularly successful in playing games such as Go [Silver et al., 2016], and Atari [Mnih et al., 2015] to a super-human level by adapting a classic RL algorithm to be used with function approximators, such as neural networks, spawning a Deep Reinforcement Learning (DRL) domain. The Deep Q-network (DQN) [Mnih et al., 2015] reinvigorated research into RL by addressing the unstable and divergent property of the action-value (Q) function when using a non-linear function approximator. This was addressed by using CNN's with experience replay [Lin, 1992] and adding a second 'target' network. This 'target' network outputs target-Q values that are used to compute the loss for every action during training in order to mitigate the network becoming destabilised by falling into a feedback loop. Following the introduction of DQN, many extensions were proposed, including Double DQN (D-DQN) [Van Hasselt et al., 2016], Prioritized experience replay [Schaul et al., 2015a], Dueling DQN [Wang et al., 2015], Bootstraped DQN [Osband et al., 2016], and DQN using optimality tightening [He et al., 2016a].

Given that DQN [Mnih et al., 2015] is a discrete-action RL algorithm, there is a need to discretise the action space in order to use this method for end-to-end manipulation. Although there has been work with that aim [James and Johns, 2016a], the problem seems naturally more suited for a solution that can operate with continuous actions. One of the first DRL continuous control algorithms was Deep Deterministic Policy Gradients DDPG [Lillicrap et al., 2015a], which is an improvement to the original DPG algorithm [Lever, 2014], by adding experience replay and target networks — much like DQN. This work lead to a vast number of continuous control algorithms for both on-policy data, such as TRPO [Schulman et al., 2015], PPO [Schulman et al., 2017], and MPO [Abdolmaleki et al., 2018], and off-policy data, such as TD3 [Fujimoto et al., 2018a] and SAC [Haarnoja et al., 2018a]. These continuous control algorithms have been used for numerous tasks over the years, including: grasping arbitrary objects [Kalashnikov et al., 2018], block stacking [Nair et al., 2018], peg-in-hole [Vecerik et al., 2017], ball-in-cup [Schwab et al., 2019], rotating a valve [Haarnoja et al., 2018b], and cloth manipulation [Matas et al., 2018]. For further reading of (pre deep) reinforcement learning applied in robotics, see [Kober et al., 2013, Kormushev et al., 2013].

1.2.3 Imitation Learning

Learning from demonstration (also known as Imitation Learning) is an attractive idea. It allows us to teach a robot how to perform a task without the need to design reward functions which can be sub-optimal and inaccurate. Moreover, it is not always desired to have a robot learn from exploration data, especially in tasks where choosing random actions could be dangerous. Imitation learning can broadly be classified into two key areas: (1) *behaviour cloning*, where an agent learns a mapping from observations to actions given demonstrations [Pomerleau, 1989, Ross et al., 2011], and (2) *inverse reinforcement learning* [Ng et al., 2000], where an agent attempts to estimate a reward function that describes the given demonstrations [Abbeel and Ng, 2004, Finn et al., 2016].

The majority of work in behaviour cloning operates on a set of configuration-space trajectories that can be collected via teleoperation [Calinon et al., 2009, Zhang et al., 2018], kinesthetic teaching [Kormushev et al., 2011, Pastor et al., 2011, Akgun et al., 2012], sensors on a human demonstrator [Dillmann, 2004, Ekvall and Kragic, 2004, Calinon and Billard, 2006, Kruger et al., 2010], through motion planners [James et al., 2017a], or even by observing humans directly. Expanding further on the latter, learning by observing humans has previously been achieved through hand-designed mappings between human actions and robot actions [Lee et al., 2013, Yang et al., 2015, Rothfuss et al., 2018], visual activity recognition and explicit hand tracking [Lee and Ryoo, 2017, Ramirez-Amaro et al., 2017], and more recently by a system that infers actions from a single video of a human via an end-to-end trained system [Yu et al., 2018].

We have barely scratched the surface of the abundance of work that now falls under the domain of end-to-end manipulation. Here we have briefly discussed two key areas in the literature today: reinforcement learning and imitation learning. Of course, the question still remains about how both of these data-driven methods compare to the pipelines that we discussed in the beginning of this section. The reality is that it is difficult to compare these two paradigms, given that manipulation pipelines are usually designed with one task in mind, while end-to-end methods are designed with the aim of solving many different tasks with the same method. This means that more often that not, a well designed pipeline should always outperform an end-to-end method, given that they are designed on a per-task basis. It suggests then, that perhaps the metric used to compare end-to-end and pipelined methods should not solely rest on a task-completion metric, but also on a time-spent metric that measures the amount of time that went into designing, implementing, training, and testing the system. This thesis does not aim to compare these two paradigms, but instead investigates a sample of methods, which fall somewhere on a spectrum from pipelined to fully end-to-end, which we believe to be more advantageous for developing a general manipulation system; one that could eventually be used in highly dynamic and unpredictable household environments.

1.3 Publications

The work described in this thesis resulted in the following publications:

- James, S. and Davison, A. J. (2021), **Attention-driven Robotic Manipulation**. *Under Review*. [James and Davison, 2021].

- James, S., Ma, Z., Arrojo, D., and Davison, A. J. (2020), **RLBench: The Robot Learning Benchmark & Learning Environment**. In *IEEE Robotics and Automation Letters (RAL) with presentation at the IEEE International Conference on Robotics and Automation (ICRA)*. [James et al., 2020].

  Video: `https://www.youtube.com/watch?v=F2PqREHT3F8`

  Code: `https://github.com/stepjam/RLBench`

- James, S., Freese, M., and Davison, A. J. (2019), **PyRep: Bringing V-REP to Deep Robot Learning**. In *CoRR*, arXiv.org. [James et al., 2019a].
  Code: https://github.com/stepjam/PyRep

- James, S., Bloesch, M., and Davison, A. J. (2018), **Task-embedded Control Networks for Few-shot Imitation Learning**. In *Proceedings of the Conference on Robot Learning (CoRL) (Oral & Best Paper Finalist)*. [James et al., 2018].
  Video: https://www.youtube.com/watch?v=iCdTo_1mHy8
  Code: https://github.com/stepjam/TecNets

- James, S., Davison, A. J., and Johns, E. (2017), **Transferring End-to-end Visuomotor Control from Simulation to Real World for a Multi-stage Task**. In *Proceedings of the Conference on Robot Learning (CoRL)*. [James et al., 2017a].
  Video: https://www.youtube.com/watch?v=k7dom7fRb8I

The following publications were co-supervised by myself and Prof. Andrew Davison during the course of this thesis:

- Bonardi, A., James, S., and Davison, A. J. (2020), **Learning One-shot Imitation from Humans Without Humans**. In *IEEE Robotics and Automation Letters (RAL) with presentation at the IEEE International Conference on Robotics and Automation (ICRA)*. [Bonardi et al., 2020].
  Video: https://www.youtube.com/watch?v=ur40SLN0_AQ
  The work in this paper was led by Alessandro Bonardi, an undergraduate student at Imperial College London. Implementation and experimentation were contributed by Alessandro Bonardi, while I contributed the method formulation, learning framework, and supervision.

- Matas, J., James, S., and Davison, A. J. (2018), **Sim-to-real Reinforcement Learning for Deformable Object Manipulation**. In *Proceedings of the Conference on Robot Learning (CoRL)*. [Matas et al., 2018].
  Video: https://www.youtube.com/watch?v=Dr0RvX1F-YQ
  Code: https://github.com/JanMatas/Rainbow_ddpg
  The work in paper was led by Jan Matas, an undergraduate student at Imperial College London. Experimentation and simulation modifications were contributed by Jan Matas, while I contributed the method formulation, learning framework, and supervision.

While not described directly, the following publications were done in conjunction with this thesis:

- James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019), **Sim-to-Real via Sim-to-Sim: Data-efficient Robotic Grasping via Randomized-to-Canonical Adaptation Networks**. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [James et al., 2019b].

1.4 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2 introduces the manipulation hardware used, the simulation stack used for learning control policies, and finally a brief overview of deep neural networks and reinforcement learning.

Chapter 3 explores some of the first work in the domain of transferring end-to-end controllers from simulation to the real world. We explore using imitation learning to learn a long-horizon multi-stage task that is analogous to a simple tidying task, and involves locating a cube, reaching, grasping, and locating a basket to drop the cube in.

Chapter 4 continues to explore transferring end-to-end controllers from simulation to the real world. In this case, we explore using a combination of state-of-the-art deep reinforcement learning algorithms to learn how to manipulating deformable objects, which includes folding a towel up to a mark, folding a face towel diagonally, and draping a piece of cloth over a hanger.

Chapter 5 introduces Task-Embedded Control Networks, which employs ideas from metric learning in order to create a task embedding that can be used by a robot to learn new tasks from one or more teleoperated demonstrations, or by observing videos of humans performing the task. Importantly, all of this is learned in simulation, without the need for expensive real-world data collection during training.

Chapter 6 presents RLBench: a challenging new benchmark and learning-environment for robot learning. The benchmark features 100 completely unique, hand-designed tasks, with the aim to facilitate research in a number of vision-guided manipulation research areas, including: reinforcement learning, imitation learning, multi-task learning, geometric computer vision, and few-shot learning.

Chapter 7 presents our Attention-driven Robotic Manipulation (ARM) algorithm, which is a general manipulation algorithm that can be applied to a range of real-world sparse-rewarded tasks without any prior task knowledge. ARM splits the complex task of manipulation into a 3 stage pipeline: (1) a Q-attention agent extracts interesting pixel locations from RGB and point cloud inputs, (2) a next-best pose agent that accepts crops from the Q-attention agent and outputs poses, and (3) a control agent that takes the goal pose and outputs joint actions. We show that ARM is successful on a range of RLBench tasks, whilst current state-of-the-art reinforcement learning algorithms catastrophically fail.

Chapter 8 concludes the thesis with a discussion of the research presented and suggestions for future work.

CHAPTER 2

Preliminaries

Contents

This chapter introduces the hardware, simulation platform, and background concepts that are used throughout this thesis.

2.1 Hardware

In this thesis, we predominantly work on 2 robotic platforms: the *Kinova Mico* and the *Franka Emika Panda*, both shown in Figure 2.1. We will briefly introduce these two arms below.

Kinova Mico

The Mico is a 6DoF arm, with a maximum reach of $70cm$, and total weight of $4.6kg$. Kinova has a library of C++ functions to control its lineup of various robots, and is referred to as the Kinova API. The API (.dll files and .h files) can be downloaded from Kinova's website as part of the Kinova software development kit (SDK). The SDK is supplemented by HTML-based documentation detailing all the available functions. The Kinova API is supported on both Windows and Ubuntu. Kinova also offers developers the possibility for developers to control the robot through a ROS interface.

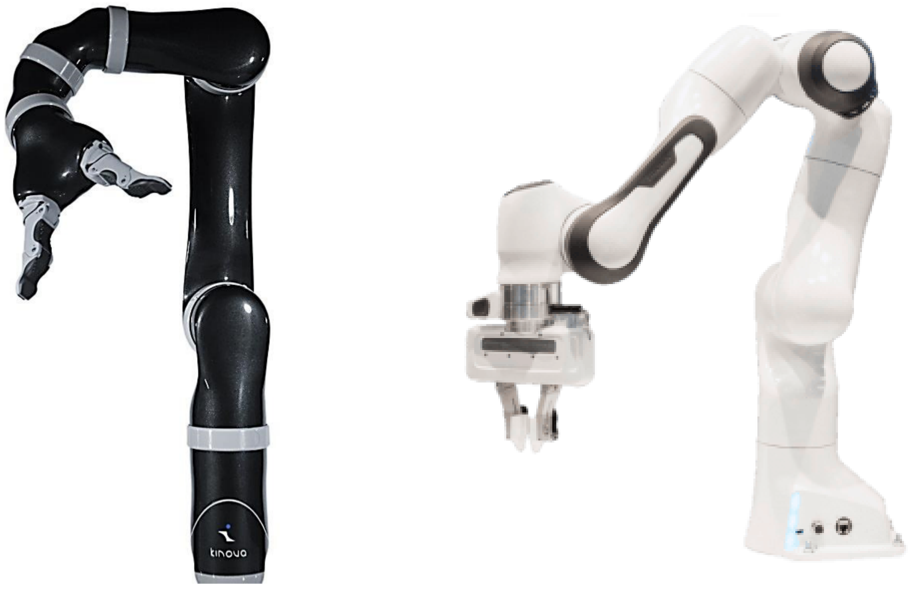


Figure 2.1: Kinova Mico (left) and Franka Emika Panda (right).

Franka Emika Panda

The Franka Emika Panda is a 7DoF arm with torque sensors in all 7 axes, a maximum reach of $85.5cm$, and a total weight of $18kg$. The franka can be controlled via the Franka Control Interface (FCI), which consists of: *libfranka*, a C++ library that provides low-level control of the robot, and *franka_ros*, the ROS integration, which includes support for ROS Control and MoveIt. The ROS integration also contains *franka_description*, a collection of URDF models and 3D meshes that can be useful outside of ROS.

2.2 Simulation Platform

It was highlighted in Section 1.1.1 that in recent years, deep learning has significantly impacted numerous areas in machine learning, improving state-of-the-art results in tasks such as image recognition, speech recognition, and language translation. Robotics has benefited greatly from this progress, with many robotics systems opting to use deep learning in many or all of the processing stages of a typical robotics pipeline [Zeng et al., 2018b, Morrison et al., 2018]. As we aim to endow robots with the ability to operate in complex and dynamic worlds, it becomes important to collect a rich variety of data of robots acting in these worlds. If we are to use deep learning however, it comes at a cost of requiring large amounts of training data, which can be particularly time consuming to collect in these dynamic environments. Simulations then, can help in one of two primary ways:

- Rapid prototyping of learning algorithms in the hope to find data-efficient solutions that can be trained on small real-world datasets that are feasible to collect.

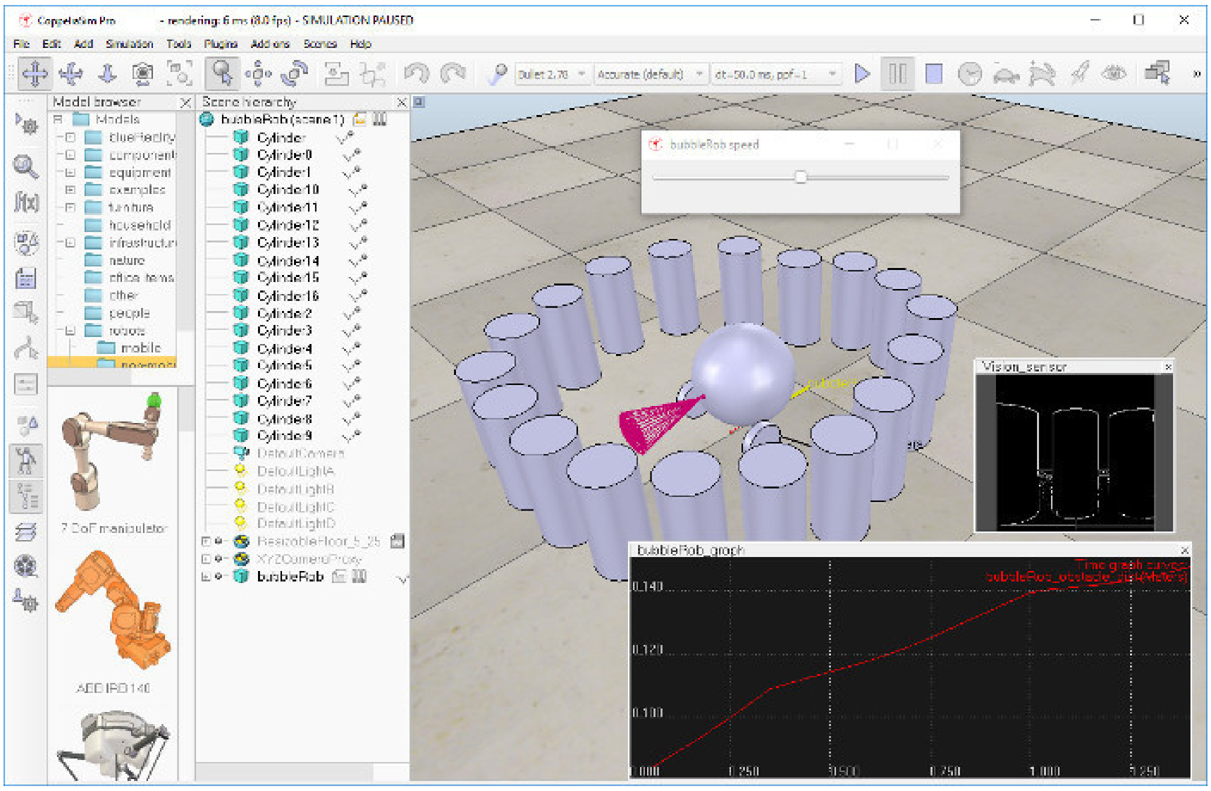


Figure 2.2: A scene from CoppeliaSim [Rohmer et al., 2013].

- Training on a large amount of simulation data with potentially a small amount of real-world data, and find ways of transferring this knowledge from simulation to the real world [James et al., 2019b, Tobin et al., 2017a, James et al., 2017a, Matas et al., 2018].

Two common simulation environments in the literature today are Bullet [Coumans, 2015] and MuJoCo [Todorov et al., 2012a]. However, given that these are physics engines rather than robotics frameworks, it can often be cumbersome to build rich environments and integrate standard robotics tooling such as inverse & forward kinematics, user interfaces, motion libraries, and path planners.

Fortunately, CoppeliaSim (formerly called V-REP) [Rohmer et al., 2013] is a general-purpose robot simulation framework maintained by *Coppelia Robotics*. An example scene from CoppeliaSim can be seen in Figure 2.2. Some of its many features include:

- Cross-platform content (Linux, Mac, and Windows).

- Several means of communication with the framework (including embedded Lua scripts, C++ plugins, remote APIs in 6 languages, ROS, etc).

- Support for 4 physics engines (Bullet, ODE, Newton, and Vortex), with the ability to quickly switch from one engine to the other.

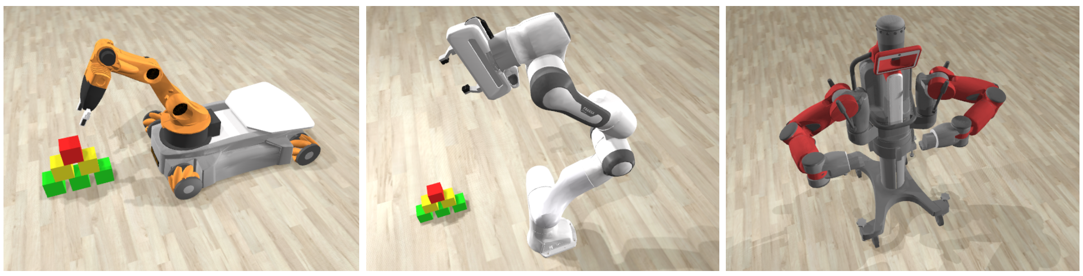


Figure 2.3: Example images of environments using the new renderer.

- Inverse & forward kinematics.
- Motion planning.
- Distributed control architecture based on embedded Lua scripts.

It is these features that make CoppeliaSim an appealing choice to use as our standard simulation platform. Python and C++ are primary languages for research in deep learning and robotics, and so it is imperative that communication times between a learning framework and CoppeliaSim are kept to a minimum. Given that CoppeliaSim was introduced in 2013 when deep learning was in its infancy, prioritisation was not given to rapid external API calls, which currently rely on inter-thread communication. As a result, this makes CoppeliaSim slow to use for external data-hungry applications.

2.3 PyRep

Although CoppeliaSim is highly customisable and ships with several API, including a Python remote API, it was not developed with the intention to be used for large-scale data collection. As a result, CoppeliaSim, when accessed via Python, is currently too slow for the rapid environment interaction that is needed in many robot learning methods, such as reinforcement learning (RL). To that end, we developed PyRep[1], as an attempt to bring the power of CoppeliaSim to the robot learning community. In addition to a new intuitive Python API and rendering engine, we modify the open-source version of CoppeliaSim to tailor it towards communicating with Python; as a result, we achieve large speed boosts in comparison of the original CoppeliaSim Python API.

Modifications

Below we outline the modifications that were made to CoppeliaSim.

[1]https://github.com/stepjam/PyRep

```
from pyrep import PyRep
from pyrep.objects import VisionSensor, Shape
from pyrep.arms import Franka

pr = PyRep()
pr.launch('my_scene.ttt', headless=True)   # Launch CoppeliaSim in a headless
   window
pr.start()  # Start the physics simulation

# Grab robot and scene objects
franka = Franka()
camera = VisionSensor('my_camera')
target = Shape('target')

while training:
    target.set_position(np.random.uniform(-1.0, 1.0, size=3))
    episode_done = False
    while not episode_done:
        # Capture observations from the vision sensor
        rgb_obs = camera.capture_rgb()
        depth_obs = camera.capture_depth()
        action = agent.act([rgb_obs, depth_obs])  # Neural network predicting
   actions
        franka.set_target_joint_velocities(action)  # Send actions to the robot
        pr.step()  # Step the physics simulation
        # Check if the agent has reached the target
        episode_done = target.get_position() == franka.get_tip().get_position()
```

Figure 2.4: PyRep API Example. Many more examples can be seen on the GitHub page.

Speed. The 6 remote APIs offered suffer from 2 communication delays. One of these comes from the socket communication between the remote API and the simulation environment (though this can be decreased considerably using shared memory). The second delay, and most notable, is inter-thread communication between the main thread and the various communication threads. This communication latency can become noticeable when the environment needs to be queried synchronously at each timestep (which is often the case in RL). To remove these latencies, we have modified the open-source version of CoppeliaSim such that Python now has direct control of the simulation loop, meaning that commands sent from Python are directly executed on the same thread. With these modifications we were able to collect robot trajectories/episodes over 4 orders of magnitude faster than using the original remote Python API; making PyRep an attractive platform for evaluation of robot learning methods.

Renderer. CoppeliaSim ships with 2 main renderers: a default OpenGL 2.0 renderer, and the POV-Ray ray tracing renderer. POV-Ray produces high quality images but at a very low framerate. The OpenGL 2.0 renderer on the other hand uses basic shadow-free rendering, and uses the old-style fixed-function pipeline OpenGL. We therefore release a new OpenGL 3.0+ renderer which supports shadow rendering from all CoppeliaSim supported lights, including directional, spotlight, and pointlight. Examples renderings can be seen in Figure 2.3.

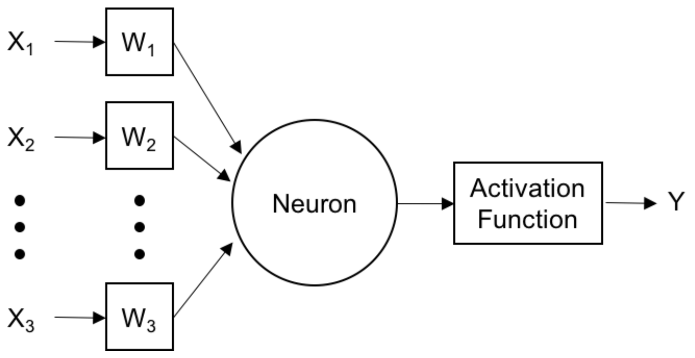


Figure 2.5: Artificial neuron model.

PyRep API. The new API manages simulation handles and provides an object-oriented way of interfacing with the simulation environment. Moreover, we have made it easy to add new robots with motion planning capabilities with only a few lines of Python code. An example of the API in use can be seen in Figure 2.4.

CoppeliaSim has been used extensively over the years in more traditional robotics research and development, but has been overlooked by the growing robot learning community. The new PyRep toolkit brings the power of CoppeliaSim to the community by providing a simple and flexible API, significant speedup in run-time, and integration of an OpenGL 3.0+ renderer to CoppeliaSim.

2.4 Deep Neural Networks

Artificial neural networks (ANNs) are quite possibly one of the greatest computational tools currently available for solving complex problems. These biologically-inspired models can be defined as a collection of densely interconnected processing units called neurons, that work in unison to perform massively parallel computations.

What makes ANNs particularly appealing is the possibility of mimicking many of the desirable characteristics of the human brain. These include massive parallelism, distributed representation and computation, learning ability, generalization ability, adaptivity, inherent contextual information processing, fault tolerance and low energy consumption [Jain et al., 1996].

The biological neuron is one of 100 billion in the human brain. Its main features consist of a cell body (soma) with branching dendrites (signal receivers) and a projection called an axon, which conduct the nerve signal. Dendrites receive electro-chemical impulses from other neurons which pass over the soma and then travel along the axon until reaching the axon terminals. Finally, the electro-chemical signal is then transmitted across the gap (synapses) between the axon terminal and a receiving cell's dendrites.

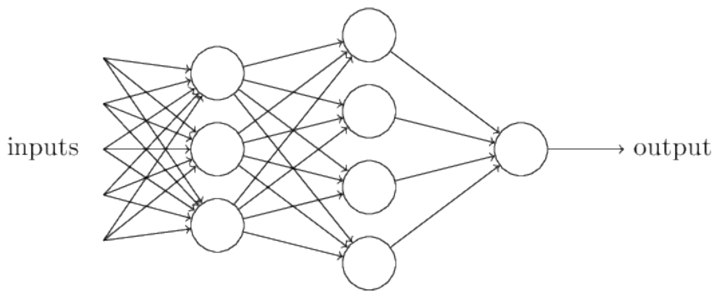


Figure 2.6: A 2 layer neural network.

Due to the large number of dendrites and synapses, it is able to receive many signals simultaneously. These signals alter the cells membrane potential which is governed by the intensity of the signal and the synaptic strength. These synapses can be adjusted by the signal passing through it, allowing them to learn the activity in which they participate.

Like its biological counterpart, the artificial neuron shown in Figure 2.5, is a model with many inputs and one output.

The neuron computes a weighted sum of its inputs to give:

$$\sum_{i=1}^{N} x_i w_i. \tag{2.1}$$

This can be seen as modelling the function of the dendrites. The soma is modelled via an activation function that determines when the neuron should fire. Common activation functions are step functions, sign functions, sigmoid function, and linear functions, as well as more recent ones like rectified linear functions.

Neurons are arranged in a weighted directed graph with connections between the inputs and outputs. The two most common networks are feed-forward networks — which contain no loops, and recurrent networks — which contain feedback connections causing loops. The networks are organized according to layers, where each neuron's output in one layer is connected to every neuron's input in the next layer. Between the input and output layers exist the hidden layers, as illustrated in Figure 2.6 — note that input layers do not count as a layer. These hidden nodes do not directly receive inputs or send outputs to the external environment.

Gradient Descent

Gradient descent is a first-order optimisation algorithm and one of the most common ways to optimise neural networks. It is a way to find local minimum of a function by starting with an initial set of values and then iteratively stepping the solution in the negative direction of the gradient until it eventually converges to zero. We define such a function as a *loss function*.

Consider a loss function $F(w)$ with parameters w, each iteration we wish to update the weights in the direction that reduces the loss in order to minimise F. Taking the gradient of the curve will point to an increase in F, so we instead negate the inverse to lower the function value:

$$w_{i+1} = w_i - \eta \frac{\partial F}{\partial w_i}, \tag{2.2}$$

where η is the *learning rate* which determines the size of the steps taken to minimise the function. Great care must be taken when choosing a learning rate — too large will result in divergence, but too small will result in a long convergence time. Below we discuss three variants of gradient descent which differ in the amount of data needed to compute the gradient of the loss function. A tradeoff must be made depending on the amount of data available and time taken to perform an update.

Stochastic gradient descent performs weight updates after the presentation of each training example. Through these frequent updates, local gradients are used to determine the direction to step, resulting in a fluctuating loss function that will on average move in the direction of the true gradient [Wilson and Martinez, 2003]. Stochastic gradient descent tends to be computationally fast due to its ability to hold its data in RAM. Moreover, it has the ability to follow curves in the error surface throughout each epoch, allowing the use of a larger learning rate which results in less iterations through training data.

Batch gradient descent uses the entire training dataset to compute the gradient of the loss function. This method tends to be slower than stochastic gradient descent, especially with large training sets where it is often orders of maginitude slower [Wilson and Martinez, 2003].

Mini-Batch gradient descent offers a hybrid of both stochastic and batch gradient descent, performing an update every n training examples. This method reduces the variance of weight updates which results in a more stable convergence than in stochastic gradient descent. Altering the batch-size n decides whether it will behave more like its stochastic version or its batch version — setting $n = 1$ results in stochastic training, while $n = N$ results in batch training, where N is the number of items in the training dataset.

Each of the 3 methods mentioned above have their strengths and weaknesses, but ultimately a choice should be made based on the expected loss function. As batch gradient descent uses the entire dataset, it is most applicable to situations where we expect a convex or smooth loss function, where we are able to move almost directly to a local or global minimum. Both stochastic gradient descent and mini-batch

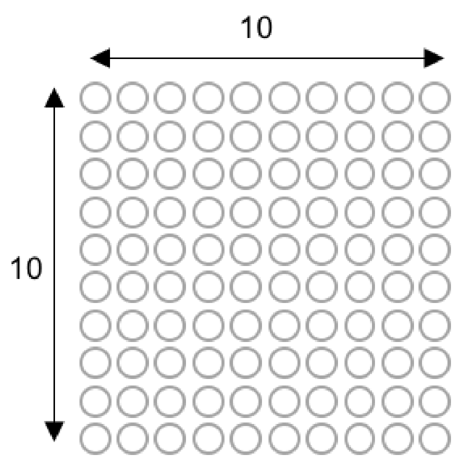


Figure 2.7: 10×10 input neurons.

gradient descent (providing n is small enough) perform better when applied to functions with many local minima. The smaller number of training samples has the ability to nudge the model out of a local minima and potentially into a more optimal minima. Mini-batch gradient descent has the capability of averaging out the noise of stochastic gradient descent by reducing the amount of jerking that is caused by single sample updates. This makes mini-batch gradient descent an ideal balance between avoiding the local minima and finding the global minima.

Deep Convolutional Neural Networks

Convolutional neural networks (CNNs) [LeCun et al., 1998] use many of the same ideas of neural networks and have shown to eliminate the need for hand-crafted feature extraction in image analysis. CNN's differ in that of fully-connected networks by taking into account the spatial structure of images — meaning that pixels which are far apart are treated differently than pixels that are close together. Today, deep convolutional networks, or some close variant, are used in most computer vision applications.

The architecture of CNNs are best explained through an example. A 10×10 image is a grid of pixel intensity values. In a CNN, we think of these as a grid of 10×10 neurons, as in Figure 2.7

In a fully-connected network, this would be treated as a network with 100 (10×10) input neurons, with every neuron in the network being connected to every neuron in an adjacent layer. As mentioned above, this has the disadvantage that the spatial structure of the images is not represented. Instead, we only make connections in small, localized regions of the input image — called the *receptive field*. This can be treated as a small window, where each of the neurons in the receptive field is connected to one hidden neuron.

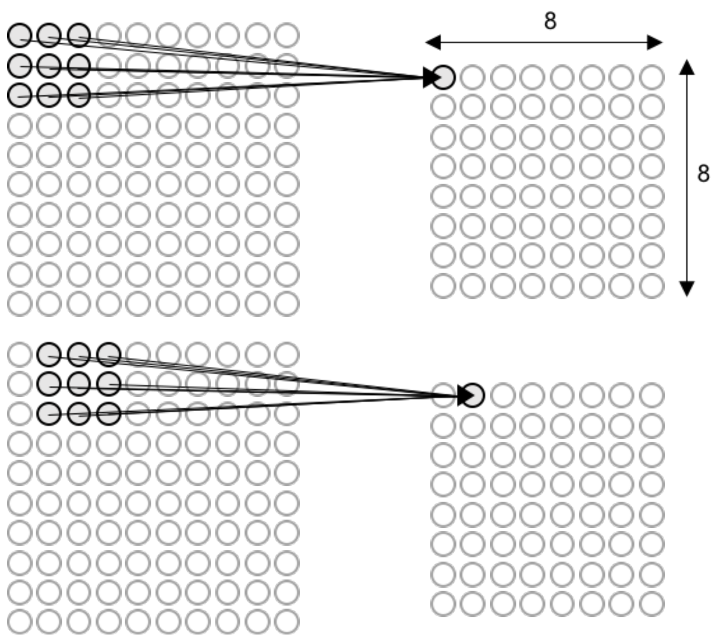


Figure 2.8: 3×3 receptive fields.

The receptive field is slid across the input image with each position of the receptive field mapping to a different hidden neuron. This continues until hitting the bottom right of the image. The amount the receptive field is slid each time is defined by the *stride length*. Figure 2.8 shows an example of a 3×3 receptive field with a stride length of 1.

With a 10×10 input image and a 3×3 receptive field, we get an 8×8 hidden layer. Just like a fully-connected network, each of the connections has an associated weight, meaning that each of the hidden neurons in an CNN has 9 (3×3) weights (and a bias) connected to it from its corresponding receptive field. Unlike a fully-connected network, these weights and bias will be the same for each of the hidden neurons, giving rise to the names *shared weights* and *shared bias*. The shared weights and bias are often said to define a *kernel* or *filter*. This gives each hidden neuron an output of:

$$\sigma(b + \sum_{i=1}^{3}\sum_{j=1}^{3} w_{i,j} a_{k+i,l+j}), \tag{2.3}$$

where σ is the activation function, b is the shared bias, $w_{i,j}$ is the shared weights of the 3 x 3 receptive field, and $a_{x,y}$ is the input activation at position (x, y).

Keeping the weights and bias the same means that the neurons in the hidden layer detect the same features at different locations in the image. This brings us to one of the fundamental properties of CNNs —*feature maps*. Feature maps are the map from one layer to the next layer, allowing us to learn features from the image instead of defining features manually.

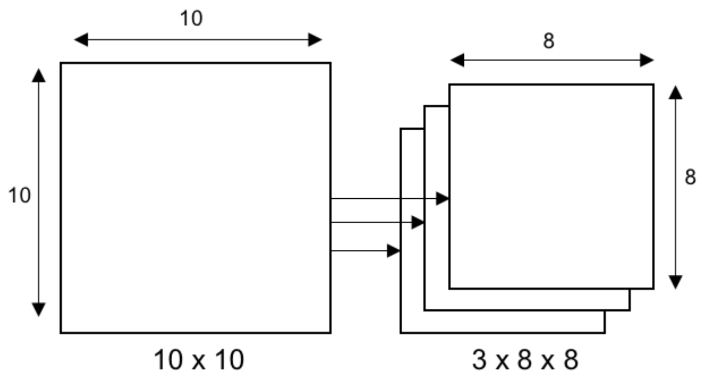


Figure 2.9: Multiple feature maps.

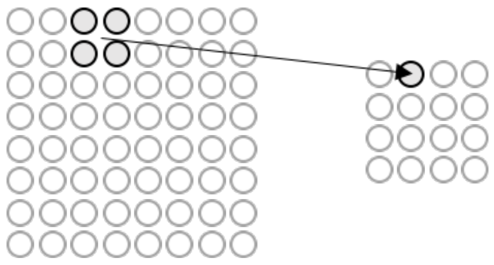


Figure 2.10: 2×2 pooling layer.

Generally we would want to extract more than one feature from an image, requiring more feature maps. Thus, a convolutional layer itself consists of several feature maps, as shown in Figure 2.9.

When sharing weights and biases we get a reduction in the number of parameters involved in a CNN in comparison to a fully-connected network. In our example, each feature map needs 9 (3×3) shared weights and a shared bias, giving a total of 10 parameters for each feature map. Suppose we use 12 feature maps giving a total of 120 (12×10) parameters for the first convolutional layer. Now suppose using a fully-connected network with 100 (10×10) input neurons representing the image, and 30 hidden neurons. With a fully connected first layer we would end up with 3,000 (30×100) weights, with an additional 30 biases, giving a total of 3,030 parameters — over 25 times the number for a CNN. This allows for faster training and construction of deep convolutional networks.

Pooling layers usually follow the convolutional layers and are responsible for taking each of the feature map's outputs to create a condensed feature map. These pooling layers are similar to the convolutional layers except that they take input from the hidden layers and summarise a region of pixels, such as the 2×2 region in Figure 2.10.

One example of a pooling layer is max-pooling, which involves taking the maximum activation in an input region (like a 2×2 region in Figure 2.10). Max-pooling is useful in highlighting whether a feature is found anywhere in a region of the image. As with the convolutional layers, usually there are many

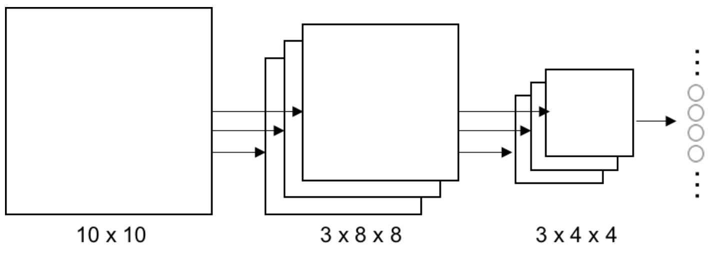


Figure 2.11: Convolutional layer, followed by a max pooling layer, and finally a fully connected layer.

feature maps, and so max-pooling is applied to each feature map separately.

The convolutional network usually ends with at least one fully connected layer. Every neuron from the final pooling layer is connected to every one of the fully connected layer — which could be the output layer at this point. Adding a pooling layer followed by a fully connected output layer to our ongoing example gives the final CNN in Figure 2.11

In a typical network, there may be several layers of convolution and pooling before the final fully connected layers.

Overall, the performance of computer vision systems depends significantly on implementation details. Generally, deeper networks perform better than shallow networks at a cost of more data and increased complexity of learning. Having said that, the dimensionality of the output layer can sometimes be reduced significantly without adverse effects on performance, as pointed out in a paper that found reducing the output layer from 4096 to 2048 actually resulted in a marginal performance boost [Chatfield et al., 2014]. Tradeoffs must be made when deciding on the size of a filter. Generally small filters are used to allow for capturing very fine details of an image and to preserve the spatial resolution, while choosing large filters could miss out on these finer details.

CNNs are used extensively throughout this thesis. Please see [O'Shea and Nash, 2015] for further reading.

2.5 Reinforcement Learning

Reinforcement learning [Sutton and Barto, 1998] falls under the wider field of machine learning. Inspired by behavioural psychology, it allows an *agent* — the learner and decision maker, to autonomously discover optimal behaviour through trial and error interactions with its surrounding environment in an attempt to solve the problems of control. The *environment* is defined as everything outside of the agent that can be interacted with, while a learning *task* is the complete specification of the environment.

In a given time step t, both the agent and environment can be modelled as being in a state $s \in \mathcal{S}$, which contain all relevant information about the current situation, e.g. a position in a navigation task. From this state, action $a \in \mathcal{A}$ can be performed. Both s and a can be members of either discrete or continuous sets. Upon advancing to the next time step, the agent receives a reward $r \in \mathcal{R}$, and transfers to the next state. A mapping from states to actions is given by a policy π. Policies can be deterministic — where the exact same action is used for a given state, or probabilistic — where the action is chosen through drawing a sample from a distribution over actions for a given state. The reinforcement learning framework defined above can be summarised in Figure 2.12.

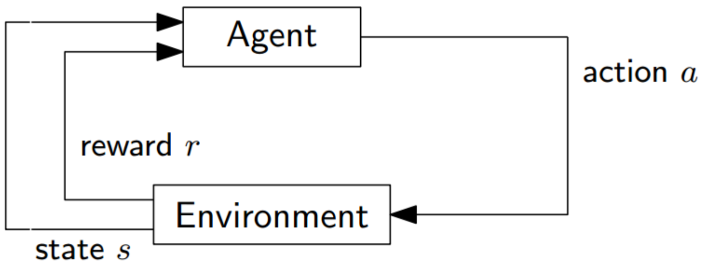


Figure 2.12: The reinforcement learning framework.

Within a task, learning is split into *episodes*. An episodes consists of trajectory τ, which is a sequence of states and actions:

$$\tau = (s_0, a_0, ..., s_T, a_T). \tag{2.4}$$

These trajectories depend on the state-transition dynamics of the environment; giving us the next state when given the current state and action: $s_{t+1} \sim P(\cdot|s_t, a_t)$.

Return

So far we have mentioned that upon advancing to the next time step, the agent receives a reward. The agent's goal is to maximise the cumulative rewards (*return*) over a trajectory:

$$R(\tau) = r_0 + r_1 + ... + r_T = \sum_{t=0}^{T} r_t. \tag{2.5}$$

where T is the final timestep. Here the final timestep is a natural break point at the end of a sequence which puts the agent in a special state called the *terminal state* — signalling the end of an *episode*. Tasks such as this are called *episodic tasks*. Following this, the agent is reset to a standard starting state or a state sampled from a standard distribution of starting states (i.e. $s_0 \sim \rho_0(\cdot)$, where ρ_0 is the start-state distribution). Conversely, we have *continuing tasks* where the agent-environment interactions go on continuously without any natural breaks (i.e. $T = \infty$).

As we can see, the return as defined in Equation (2.5) may not be suitable for all tasks, as rewards gained now are worth just as much as rewards gained in the future. This brings us to the concept of *discounting*, where we introduce a parameter γ (the *discount rate*) that ranges $0 \leq \gamma \leq 1$. This is used to determine the present value of future rewards the agent may receive. Adding the discount rate to our return, gives us the new *discounted return*:

$$R(\tau) = r_0 + \gamma r_1 + \gamma^2 r_2 + ... = \sum_{t=0}^{T} \gamma^t r_t. \tag{2.6}$$

Great care must be taken when choosing an appropriate value for γ as it often qualitatively changes the form of the optimal solution [Kaelbling et al., 1996]. As γ approaches 0, the agent becomes myopic, only choosing actions that maximise its immediate reward, leading to poor performance in long term tasks. On the contrary, as γ approaches 1, the agent becomes more far-sighted, leading to the problem that it cannot distinguish between policies that immediately gain a large amount of reward, and those that gain a reward in the future.

Model-Free Reinforcement Learning

Reinforcement learning is a large field, and at the highest level of abstraction can be broken into two main areas: model-based and model-free. Model-based approaches use a predictive model of the world (either given or learned) which can be used to query and plan, while model-free approaches forgoes the model and learns a policy directly on observed data from the environment. In robot manipulation, a ground-truth model of the world and its dynamics is not available, and so usually this requires a model to be learned from experience, which itself can be a challenge. For this reason, model-free algorithms are particularly popular for learning end-to-end control policies for manipulation. This thesis does not contain any model-based contributions, and so the remaining sections will focus on model-free approaches.

Model-free methid can be broken down into 3 core areas: value-based methods, policy gradient methods, and actor-critic methods. Below, we briefly describe each of these.

Value-based Methods

When an agent enters into a new state, it is beneficial for it know how valuable it is to be in this new state. The value of a state can be measured in two ways — *state-value functions* and *action-value functions*.

The *state-value function* for a policy π is the expected return when starting in a state s and following π thereafter and is defined as:

$$V^{\pi}(s) = E_{\tau\sim\pi}[R(\tau)|s_t = s] = E_{\tau\sim\pi}[\sum_{t=0}^{T}\gamma^k r_t|s_t = s], \tag{2.7}$$

where $E_{\tau\sim\pi}$ described the expected value of following policy π.

The *action-value function* for a policy π, is the expected return when starting in state s, taking action a, and following π thereafter, and is defined as:

$$Q^{\pi}(s,a) = E_{\tau\sim\pi}[R(\tau)|s_t = s, a_t = a] = E_{\tau\sim\pi}[\sum_{t=0}^{T}\gamma^t r_t|s_t = s, a_t = a]. \tag{2.8}$$

The *action-value function*, or Q-function, brings us to one of the most widely used value-based methods: Q-learning. Q-learning [Watkins, 1989] is an temporal-difference (TD) algorithm; where *Temporal-difference (TD) learning* is a combination of Dynamic Programming (DP) — the ability to learn through bootstrapping, and Monte Carlo (MC) — the ability to learn directly from samples taken from the environment without access to the Markov Decision Process (MDP). Q-learning is also an off-policy algorithm, and so directly approximates Q independent of the policy being followed.

In Q-learning we aim to estimate the optimal Q-function $Q^*(s,a)$, which is defined to be the maximum return that can be obtained by starting at state s, taking action a, and following the optimal policy thereafter. The optimal Q-function obeys the Bellman optimality equation:

$$Q^*(s_t, a_t) = E[r + \gamma \max_a Q^*(s_t, a)]. \tag{2.9}$$

.

We can learn the Q function by using *temporal errors* to inform us how different the new value is from the old prediction. Given an experience (s, a, r, s'), in which the agent starts in state s, performs action a, receives a reward r, and transfers to state s', we can then define the update to Q as:

$$Q_{i+1}(s_t, a_t) \leftarrow Q_i(s_t, a_t) + \alpha[r + \gamma \max_a Q_i(s_{t+1}, a) - Q_i(s_t, a_t)]. \tag{2.10}$$

By definition of the Bellman optimality equation, it has been shown that the above equation converges to the optimal Q-function as i approaches infinity: $Q_i \to Q^*$ as $i \to \infty$.

Historically, value functions were represented as tables with one entry for each state. The question is: what happens when we are given a large, or even continuous state space, such as the case with our robot arm? With a continuous state space, comes large, memory consuming tables that require a large amount of time and data. Therefore, the only way to learn in situations like these, is to generalise from previous states to ones that we have not seen before. This motivates the need for function approximation methods, such as Deep Q-Learning [Mnih et al., 2015], which is discussed further and expanded upon in Section 7.3.

Policy Gradient Methods

Policy gradient methods directly optimises a policy $\pi_\theta(a|s)$ without the need for a value function by optimising the parameters θ directly via gradient ascent on a policy objective $J(\pi_\theta)$:

$$\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\pi_\theta). \tag{2.11}$$

.

Here, the gradient of the policy objective $\nabla_\theta J(\pi_\theta)$ is called the *policy gradient*. To compute this analytically, we assume π_θ to be differential and that we know the gradient of the policy. Below, we show the derivation of one of the simplest forms of the gradient. Recall that the goal of the agent is to maximise the expected reward. We therefore define the objective function as: $J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$, and retrieve the policy gradient as follows:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta E_{\tau\sim\pi_\theta}[R(\tau)] \\
&= \int_\tau \nabla_\theta \pi_\theta(\tau) R(\tau) \\
&= \int_\tau \nabla_\theta \log \pi_\theta(\tau) R(\tau) \\
&= E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s|a) R(\tau)],
\end{aligned} \tag{2.12}$$

where $\pi_\theta(\tau) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$, and where $\rho_0(s_0)$ represents sampling from the start-state distribution. Note that we retrieve $\nabla_\theta \pi_\theta(\tau) = \pi_\theta(\tau)\nabla_\theta \log \pi_\theta(\tau)$ via the log-derivative trick.

Due to the fact that each trajectory during training can greatly deviate from each other, policy gradient methods tend to have high variance. Consequently, this high variance will make noisy gradients, cause unstable learning, and skew the policy distribution in an undesired direction. It is for this reason that actor-critic methods exist.

Actor-Critic Methods

Actor-critic methods take the best of both value-based and policy gradient methods by using the value function (critic) as a way to inform the direction of the policy (actor) gradient. *Action-value actor-critic* can be considered one of the simplest actor-critic methods, and involves modifying the policy gradient equation to instead follow an approximate policy gradient:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(s|a) Q(s,a)], \tag{2.13}$$

Two actor-critic methods that are used in this thesis are Deep Deterministic Policy Gradients (DDPG), which is described in Section 4.3, and Soft Actor-Critic (SAC), which is described in Section 7.3. We point the reader to those sections for a description of these algorithms.

CHAPTER 3

Transferring End-to-end Controllers from Simulation to Reality

Contents

3.1 Introduction

As described in Chapter 1, an emerging trend for robot manipulation is to learn controllers directly from raw sensor data in an end-to-end manner. This is an alternative to traditional pipelined approaches which often suffer from propagation of errors between each stage of the pipeline. End-to-end approaches have had success both in the real world [Levine et al., 2016a, Montgomery and Levine, 2016, Montgomery et al., 2016] and in simulated worlds [Popov et al., 2017, James and Johns, 2016a, Zhang et al., 2015, Higgins et al., 2018]. Learning end-to-end controllers in simulation is an attractive alternative to using physical robots due to the prospect of scalable, rapid, and low-cost data collection. However, these simulation approaches are of little benefit if we are unable to transfer the knowledge to the real world. What we strive towards are robust, end-to-end controllers that are trained in simulation, and can run in the real world without having seen a single real world image.

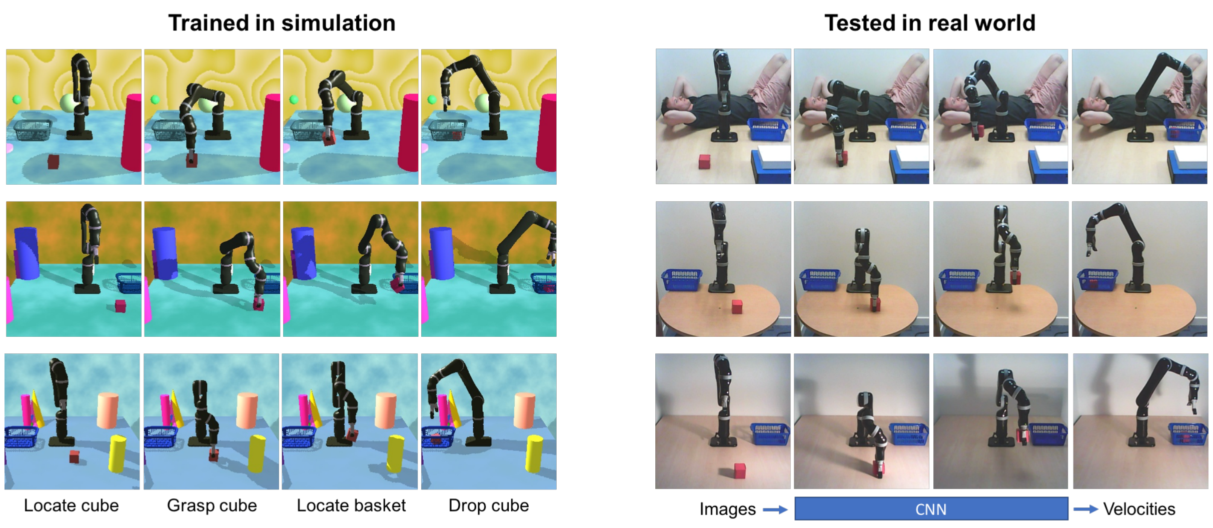


Figure 3.1: Our approach uses simulation to collect a series of control velocities to solve a multi-stage task. This is used to train a reactive neural network controller which continuously accepts images and joint angles, and outputs motor velocities. By using domain randomisation, the controller is able to run in the real world without having seen a single real image.

In this chapter, we present work that accomplishes the goal of transferring end-to-end controllers to the real world and demonstrate this by learning a long-horizon multi-stage task that is analogous to a simple tidying task, and involves locating a cube, reaching, grasping, and locating a basket to drop the cube in. This is accomplished by using demonstrations of linear paths constructed via inverse kinematics (IK) in the Cartesian space, to construct a dataset that can then be used to train a reactive neural network controller which continuously accepts images along with joint angles, and outputs motor velocities. We also show that task performances improves with the addition of auxiliary outputs (inspired by [Jaderberg et al., 2016, Dilokthanakul et al., 2017]) for positions of both the cube and gripper. Transfer is made possible by simply using domain randomisation [Sadeghi and Levine, 2016, Tobin et al., 2017a], such that through a large amount of variability in the appearance of the world, the model is able to generalise to real world environments. Figure 3.1 summarises the approach, whilst our video demonstrates the success of the final controller in the real world [1].

Our final model is not only able to run on the real world, but can accomplish the task with variations in the position of the cube, basket, camera, and initial joint angles. Moreover, the model shows robustness to distractors, lighting conditions, changes in the scene, and moving objects (including people).

Our contributions in this work are 2 fold. Firstly, we show that end-to-end control of a robot arm can be transferred to the real world through the use of procedural domain randomisation. Secondly, we perform a detailed ablation study to answer several important questions, such as: how performance

[1]Video: https://youtu.be/X3SD56hporc

varies as we alter dataset size? What is most important to randomise during domain randomisation? And how do architectural decisions affect performance?

3.2 Related Work

End-to-end methods for control are often trained using Reinforcement Learning (RL). In the past few years, classic RL algorithms have been fused with function approximators, such as neural networks, to spawn the domain of deep reinforcement learning, which is capable of playing games such as Go [Silver et al., 2016] and Atari [Mnih et al., 2015] to a super-human level. There have been advances in applying these techniques to both simulated robotic platforms [Popov et al., 2017, James and Johns, 2016a], and real-world platforms [Gu et al., 2017], but fundamental challenges remain, such as sample inefficiency, slow convergence, and the algorithm's sensitivity to hyperparameters. There have been attempts at transferring trained polices from the real world following training in simulation, but these attempts have either failed [James and Johns, 2016a], or required additional training in the real world [Rusu et al., 2017a]. Although applying RL can work well for computer games and simulations, the same cannot be said as confidently for real-world robots, where the ability to explore sufficiently can become intractable as the complexity of the task increases. To counter this, imitation learning can be used by providing demonstrations in first [Duan et al., 2017, Lee et al., 2015a, Hausman et al., 2017] or third [Stadie et al., 2017] person perspective. Our method uses the full state of the simulation to effectively produce first-person demonstrations for supervision without the need for exploration.

A different approach for end-to-end control is guided policy search (GPS) [Levine and Koltun, 2013, Levine and Abbeel, 2014], which has achieved great success particularly in robot manipulation [Levine et al., 2016a, Montgomery and Levine, 2016, Montgomery et al., 2016]. Unlike most RL methods, these approaches can be trained on real world robotic platforms, but therefore have relied on human involvement which limits their scalability. To overcome human involvement, GPS has been used with domain adaptation of both simulated and real world images to map to a common feature space for pre-training [Tzeng et al., 2016]; however, this still requires further training in the real world. In comparison, our method has never seen a real world image before, and learns a controller purely in simulation.

One approach to scale up available training data is to continuously collect data over long periods of time using one [Pinto and Gupta, 2016b] or multiple robots [Levine et al., 2016c, Finn and Levine, 2017a]. This was the case for [Levine et al., 2016c], where 14 robots were run for a period of 2 months and collected $800,000$ grasp attempts by randomly performing grasps. A similar approach was used to learn a predictive model in order to push objects to desired locations [Finn and Levine, 2017a]. Although the results are impressive, there is some doubt in scalability with the high purchase cost of robots, in addition to the question of how you would get data for more complex and long-horizon tasks. Moreover, these solutions typically cannot generalise to new environments without also training them in that same environment.

There also exist a number of works that do not specifically learn end-to-end control, but do in fact use simulation to learn behaviours with the intention to use the learned controller in the real world. Such works include [Johns et al., 2016], which uses depth images from simulated 3D objects to train a CNN to predict a score for every possible grasp pose. This can then be used to locate a grasp point on novel objects in the real world. Another example is [Christiano et al., 2016], where deep inverse models are learned within simulation to perform a back-and-forth swing of a robot arm using position control. For each time step during testing, they query the simulated control policy to decide on suitable actions to take in the real world.

Transfer learning is concerned with transferring knowledge between different tasks or scenarios. Works such as [Devin et al., 2017, Gupta et al., 2017] show skill transfer within simulation, whereas we concentrate on simulation-to-real transfer. In [Sadeghi and Levine, 2016], the focus is on learning collision-free flight in simulated indoor environments using realistic textures sampled from a dataset. They show that the trained policy can then be directly applied to the real world. Their task differs to ours in that they do not require hand-eye coordination, and do not need to deal with a structured multi-stage task. Moreover, rather than sampling from a dataset of images, ours are procedurally generated, which allows for much more diversity. During development of our work, a related paper emerged which uses the domain randomisation [Tobin et al., 2017a] method in a similar manner to us, except that the focus is on pose estimation rather than end-to-end control. We operate at the lower level of velocity control, to accomplish a multi-stage task which requires not only pose estimation, but also target reaching and grasping. In addition, we show that our learned controller can work in a series of stress tests, including scenes with dramatic illumination changes and moving distractor objects.

3.3 Approach

Our aim is to create an end-to-end reactive controller that does not require real-world data to train, and is able to learn complex behaviours in a short period of time. To achieve this, we generate a large number of trajectories in simulation, together with corresponding image observations, and then train a controller to map observed images to motor velocities, which are effected through a PID controller. Through the use of domain randomisation during the data generation phase, we are able to run the controller in the real world without having seen a single real image. We now describe in detail the dataset generation and training method.

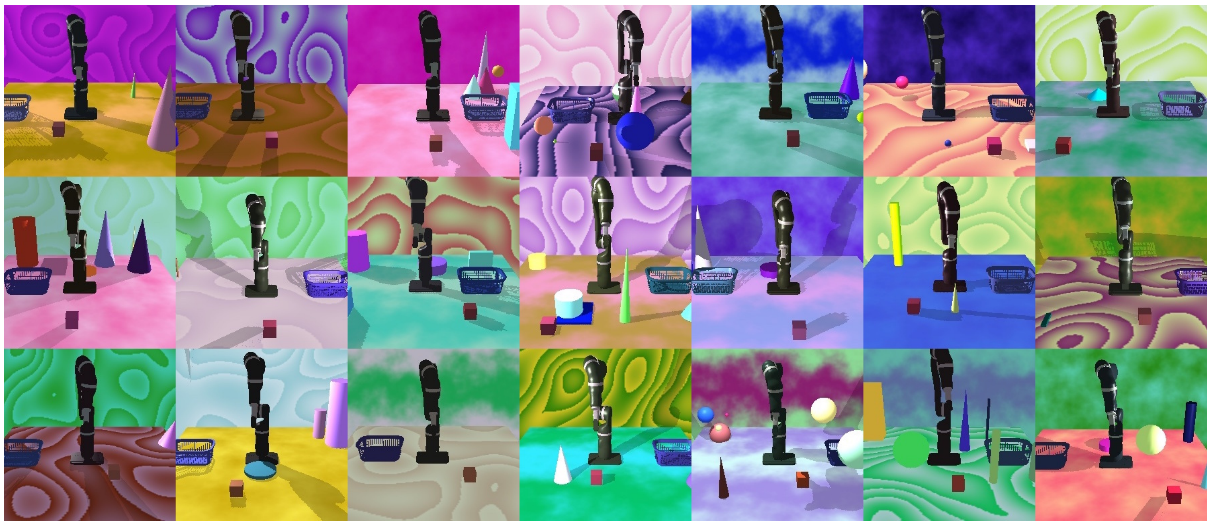


Figure 3.2: A collection of generated environments as part of our domain randomisation method.

3.3.1 Data Collection

The success of this work comes down to the way in which we generate the training data (Figure 3.2). Our approach uses a series of linear paths constructed in the Cartesian space via inverse kinematics (IK) in order to construct the task sequence. At each simulation step, we record motor velocities, joint angles, gripper actions (open or close command), cube position, gripper position, and camera images. We split the task into 5 stages, and henceforth, we refer to the sequence of stages as an *episode*. At the start of each episode, the cube and basket is placed randomly within an area that is shown in Figure 3.3. We use the V-REP [E. Rohmer, 2013] simulator during all data collection.

In the first stage of the task, the arm is reset to an initial configuration. We place a waypoint above the cube and plan a linear path which we then translate to motor velocities to execute. Once this waypoint has been reached, we execute the second stage, where by a closing action on the gripper is performed. The third stage sets a waypoint a few inches above the cube, and we plan and execute a linear path in order to lift the cube upwards. The fourth stage places a waypoint above the basket, where we plan and execute a final linear path to take the grasped cube above the basket. Finally, the fifth stage simply performs a command that opens the gripper to allow the cube to fall into the basket. A check is then carried out to ensure that the location of the cube is within the basket; if this is the case, then we save the episode. We do not consider obstacle avoidance in this task, and so episodes that cannot find a linear set of paths due to obstacles are thrown away. The data generation method can be run on multiple threads, which allows enough data for a successful model to be collected within a matter of hours, and increasing the number of threads would further reduce this time further.

Using this approach, it is already possible to train a suitable neural network to learn visuomotor control

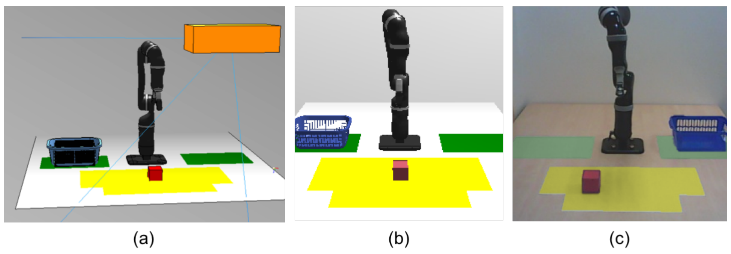


Figure 3.3: Variations in the positioning of the cube, basket, and camera, during both training and testing. In all three images, the yellow area represents the possible locations of the cube, and the green areas represent the possible locations of the basket. (a) shows variations of the camera pose, illustrated by the orange cuboid. (b) shows an example view from the camera in simulation. (c) shows an example view from the camera in the real world. Note that the yellow, green, and orange shapes are purely for visualisation here, and are not seen during training or testing.

of the arm and perform the task to succeed 100% of the time when tested in simulation. However, we are concerned with applying this knowledge to the real world so that it is able to perform equally as well as it did in simulation. By using domain randomisation, we are able to overcome the reality-gap that is present when trying to transfer from the synthetic domain to the real world domain. For each episode, we list the environment characteristics that are varied, followed by an explanation of why we vary them:

- The colour of the cube, basket, and arm components are sampled from a normal distribution, with the mean set as close to the estimated real world equivalent; though these could also be sampled uniformly.
- The position of the camera, light source (shadows), basket, and cube are sampled uniformly. Orientations are kept constant.
- The height of the arm base from the table is sampled uniformly from a small range.
- The starting joint angles are sampled from a normal distribution with the mean set to the configuration in Figure 3.3.
- We make use of Perlin noise [Perlin, 2002] composed with functions (such as sine waves) to generate textures, which are then applied to the table and background of the scene.
- We add random primitive shapes as distractors, with random colours, positions, and sizes sampled from a uniform distribution.

A selection of these generated environments can be seen in Figure 3.2. We chose to vary the colours and positions of the objects in the scene as part of a basic domain randomisation process. Slightly less obvious is varying the height of the arm base; this is to avoid the network learning a controller for a

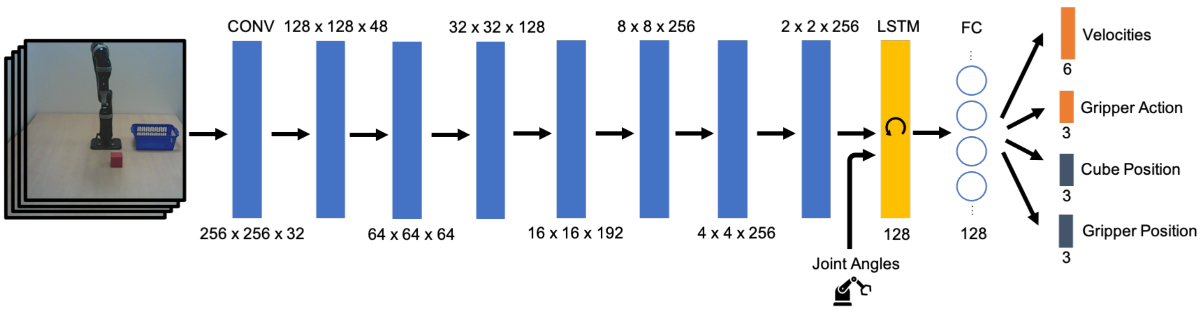


Figure 3.4: Our network architecture continuously maps sequences of the past 4 images and joint angles to motor velocities and gripper actions, in addition to 2 axillary outputs: the 3D cube position and 3D gripper position. These axillary outputs are not used during testing, but are instead present to help the network learn informative features.

set height, that we cannot guarantee is the true height in the real world. In order to successfully run these trained models in the real world, we also must account for non-visual issues, such as the error in the starting position of the joints when run on the real world. Although we could send the real world arm to the same starting position as the synthetic arm, in practice the joint angles will be slightly off, and this could be enough to put the arm in configurations that are unfamiliar and lead to compounding errors along its trajectory. It is therefore important that the position of the robot at the start of the task is perturbed during data generation. This leads to controllers that are robust to compounding errors during execution, since a small mistake on the part of the learned controller would otherwise put it into states that are outside the distribution of the training data. This method can also be applied to the generated waypoints, but in practice this was not needed. We use procedural textures rather than plain uniform textures, to achieve sufficient diversity which encompasses background textures of the real world.

Note that no dynamics randomisation is performed during data collection; this is because the task considered in this chapter does not contain any complex contact dynamics (e.g. pushing, insertion, etc), and the task can be completed without taking into account physics.

3.3.2 Network Architecture

The network, summarised in Figure 3.4, consists of 8 convolutional layers each with a kernel size of 3×3, excluding the last, which has a size of 2×2. Dimensionality reduction is performed at each convolutional layer by using a stride of 2. Following the convolutional layers, the output is concatenated with the joint angles and then fed into an LSTM module (we discuss the importance of this in the results). Finally, the data goes through a fully-connected layer of 128 neurons before heading to the output layer. We chose to output velocities, effected via a PID controller, rather than to output torques directly, due to the difficulty in transferring complex dynamics from simulation.

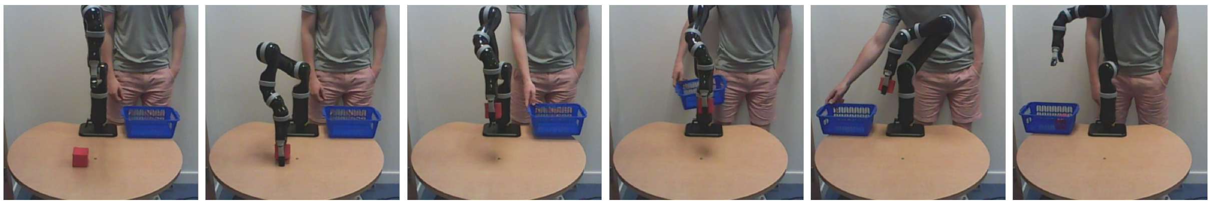


Figure 3.5: A sequence of images showing the network's ability to generalise. Whilst a person is standing in the view of the camera, the robot is able to grasp the cube and proceed to the basket. As the arm progresses towards the basket, the person moves the basket to the other side of the arm. Despite this disruption, the network alters its course and proceeds to the new location. During training, the network had never seen humans, moving baskets, or this particular table.

The network outputs 6 motor velocities, 3 gripper actions, and 2 auxiliary outputs: cube position and gripper position. We treat the 3 gripper actions as a classification problem, where the outputs are $\{open, close, no\text{-}op\}$. During testing, the auxiliary outputs are not used at any point, but are present to aid learning and conveniently help debug the network. By rendering a small marker at the same positions as the auxiliary outputs, we are able to observe where the controller estimates the cube is, in addition to where it estimates the gripper is, which can be helpful during debugging.

Our loss function $\mathcal{L}_{Total}$ is a combination of the mean squared error of the velocities ($\mathcal{L}_V$) and gripper actions ($\mathcal{L}_G$), together with the gripper position ($\mathcal{L}_{GP}$) auxiliary and cube position ($\mathcal{L}_{CP}$) auxiliary, giving

$$\mathcal{L}_{Total} = \mathcal{L}_V + \mathcal{L}_G + \mathcal{L}_{GP} + \mathcal{L}_{CP}. \tag{3.1}$$

We found that simply weighting the loss terms equally resulted in effective and stable training. The model was trained with the Adam optimiser [Kingma and Ba, 2015] with a learning rate of 10^{-4}.

3.4 Experiments

In this section we present results from a series of experiments, not only to show the success of running the trained models in different real world settings, but also to show what aspects of the domain randomisation are most important for a successful transfer. Figure 3.5 shows an example of the controller's ability to generalise to new environments in the real world, although many more are shown in the video[2]. We focused our experiments to answer the following questions:

1. How does performance vary as we alter the dataset size?

[2]Video: `https://youtu.be/X3SD56hporc`

2. How robust is the trained controller to new environments?
3. What is most important to randomise during domain randomisation?
4. Does the addition of auxiliary outputs improve performance?
5. Does the addition of joint angles as input to the network improve performance?

Experimental Setup We first define how we evaluate the controller in both the simulation and real world. We place the cube using a grid-based method, where we split the area in which the cube was trained into a grid of $10cm \times 10cm$ cells. Using the grid, the cube can be in one of 16 positions, and for each position we run a trial twice with the basket on each side of the arm, resulting in 32 trials; therefore, all of our results are expressed as a percentage based on 32 trials. This number of trails is sufficient for a trend to emerge, as shown later in Figure 3.7. In Figure 3.6, we summarise the testing conditions, where each square represents a position, and the two triangles represent whether the basket was on the left or right side of the robot at that position.

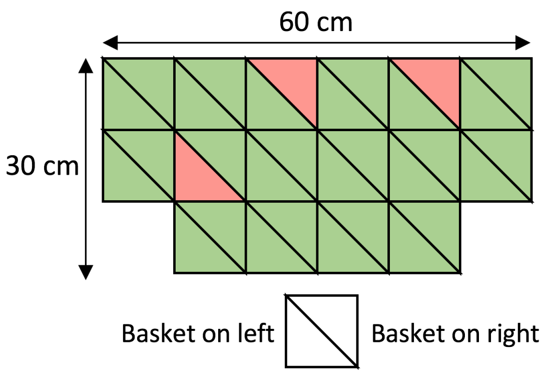


Figure 3.6: Controller evaluation example. Green signifies a success, whilst red signifies a failure. In this case, success would be 91%.

Altering Dataset Size To answer the first question of how the dataset size effects performance, we train several instances of the same network on dataset sizes ranging from 100,000 to 1 million images. Figure 3.7 shows the success rates in both simulation and real world, for the task in an environment with no distractors (such as in Figure 3.3). The graph shows that a small dataset of 200,000 images achieves good performance in simulation, but 4 times more data is needed in order to achieve approximately the same success rate in the real world. In-

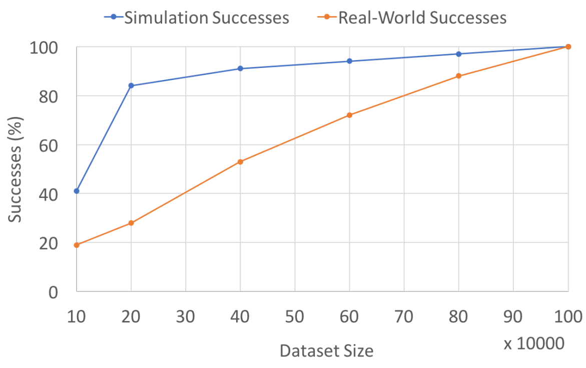


Figure 3.7: How dataset size effects performance in both simulation and real world.

terestingly, both simulation and real world achieve 100% for the first time on the same dataset size (1 million).

Robustness to New Environments We now turn our attention to Figure 3.8 and Table 3.1, which provide a summary of our results for the remaining research questions. First, we discuss the results in the top half of Table 3.1, where we evaluate how robust the network is to changes in the testing environment. Successes are categorised into *cube vicinity* (whether the gripper's tip reached within $\approx 2cm$ of the cube), *cube grasped* (whether the gripper lifted the cube off the table), and full task (whether the cube was reached, grasped, and dropped into the basket). The first two results show our solution achieving 100% when tested in both simulation and the real world. Although the network was trained with distractors, it was not able to achieve 100% with distractors in the real world. Note that the controller does not fail once the cube has been grasped, but rather fails during the reaching or grasping. The majority of failures in this case where when the cube was closest to the distractors in the scene. In the moving scene test, a person waved their arm back and forth such that each frame saw the arm in a different location. Reaching was not affected in this case, but grasping performance was. The moving camera test was performed by continuously raising and lowering the height of the tripod where the camera was mounted within a range of 2 inches during each episode. Although never experiencing a moving scene or camera motion during training, overall task success remained high at 81% and 75% respectively. To test invariance to lighting conditions, we aimed a bright spotlight at the scene and then moved the light as the robot moved. The results show that the arm was still able to recognise the cube, achieving a 84% cube vicinity success rate, but accuracy in the grasp was affected, resulting in the cube only being grasped 56% of the time. This was also the case when we replaced the cube with one that was half the length of the one seen in training. The 89% vicinity success in comparison to the 41% grasp success shows that this new object was too different to perform an accurate grasp, and the gripper would often only brush the cube. Interestingly, when the cube was replaced with a novel object such as a stapler or wallet, the task was occasionally successful. One explanation for this behaviour could be down to a clear colour discontinuity in comparison to the background in the area in which the cube is normally located.

Ablation Study The bottom half of Table 3.1 focuses on the final 3 questions regarding which aspects are important to randomise during simulation, and whether auxiliary tasks and joint angles improve performance. Firstly, we wish to set a baseline which shows that naively simulating the real world is difficult for getting high success in the real world. We generated a dataset based on a scene with colours close to the real world. The baseline is unable to succeed at the overall task, but performs well at reaching. This conclusion is in line with other work [James and Johns, 2016a, Zhang et al.,

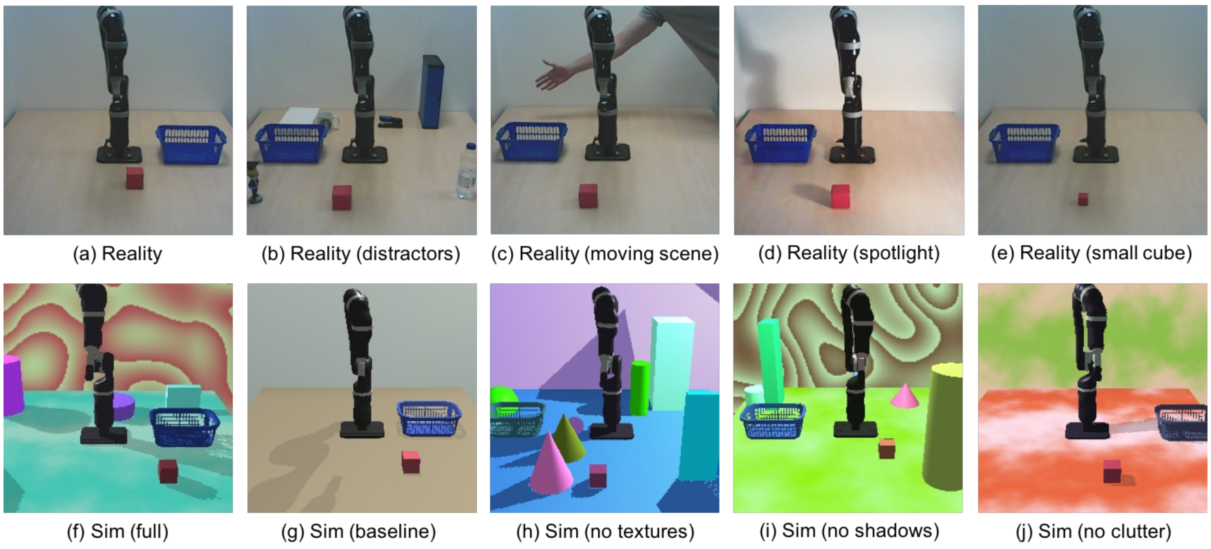


Figure 3.8: Images (a) to (e) are taken from the real world, whilst images (f) to (j) are taken from simulation. The real world images show the testing scenarios from Table 3.1, whilst the simulated images shows samples from the training set from Table 3.1.

2015]. It is clear that having no domain randomisation does not transfer well for tasks that require accurate control. Whilst observing the baseline in action, it would frequently select actions that drive the motors to force the gripper into the table upon reaching the cube. Training a network without distractors and testing without distractors yields 100%, whilst testing with distractors unsurprisingly performs poorly. A common characteristic of this network is to make no attempt at reaching the cube, and instead head directly to above the basket. We tested our hypothesis that using complex textures yields better performance than using colours drawn from a uniform distribution. Although reaching is not affected, both grasping and full task completion degrade to 69% and 44% respectively. Moreover, swapping to the table illustrated in Figure 3.5 leads to complete failure when using plain colours.

Without moving the camera during training, the full task is not able to be completed. Despite this, target reaching seems to be unaffected. We cannot guarantee the position of the camera in the real world, but the error is small enough such that the network is able to reach the cube, but large enough that it lacks the ability to achieve a grasp. Results show that shadows play an important role following the grasping stage. Without shadows, the network can easily become confused by the shadow of both the cube and its arm. Once grasped, the arm frequently raises and then lowers the cube, possibly due to the network mistaking the shadow of the cube for the cube itself.

We now observe which aspects of the network architecture contribute to success when transferring the controllers. We alter the network in 3 distinct ways – no LSTM, no auxiliary output, and no joint angles – and evaluate how performance differs. Excluding the LSTM unit from the network causes the network to fail at the full task. Our reasoning for including recurrence in the architecture was to ensure that state

Scenario		Successes		
Train	Test	Cube vicinity	Cube grasped	Full task
Sim (full)	Sim (full)	100%	100%	100%
Sim (full)	Real	100%	100%	100%
Sim (full)	Real (distractors)	89%	75%	75%
Sim (full)	Real (moving scene)	100%	89%	89%
Sim (full)	Real (moving camera)	97%	89%	75%
Sim (full)	Real (spotlight)	84%	56%	56%
Sim (full)	Real (small cube)	89%	41%	41%
Sim (baseline)	Real	72%	0%	0%
Sim (no distractors)	Real	100%	100%	100%
Sim (no distractors)	Real (distractors)	53%	0%	0%
Sim (no textures)	Real	100%	69%	44%
Sim (no moving cam)	Real	100%	9%	3%
Sim (no shadows)	Real	81%	46%	19%
Sim (no LSTM)	Real	100%	56%	0%
Sim (no auxiliary)	Real	100%	84%	84%
Sim (no joint angles)	Real	100%	44%	25%

Table 3.1: Results based on 32 trials from a dataset size of 1 million images ($\approx$ 4000 episodes) run on a square table in the real world. Successes are categorised into *cube vicinity* (whether the arm reached within $\approx 2cm$ of the cube, based on human judgement), *cube grasped*, and *full task* (whether the cube was reached, grasped and dropped into the basket). The top half of the table focuses on testing the robustness of the full method, whilst the bottom half focuses on identifying what are the key aspects that contribute to transfer. A sample of the scenarios can be seen in Figure 3.8.

was captured. As this is a multi-stage task, we felt it important for the network to know what stage of the task it was in, especially if it is unclear from the image whether the gripper is closed or not (which is often the case). Typical behaviour includes hovering above the cube, which then causes the arm to drift into unfamiliar states, or repeatedly attempting to close the gripper even after the cube has been grasped. Overall, the LSTM seems fundamental in this multi-stage task. The next component we analysed was the auxiliary outputs. The task was able to achieve good performance without the auxiliaries, but not as high as with the auxiliaries, showing the benefit of this auxiliary training. Finally, the last modification we tested was to exclude joint angles. We found that the joint angles helped significantly in keeping the gripper in the orientation that was seen during training. Excluding the joint angles often led to the arm reaching the cube vicinity, but failing to be precise enough to grasp. This could be because mistakes by the network are easier to notice in joint space than in image space, and so velocity corrections can be made more quickly before reaching the cube.

3.5 Conclusion

In this chapter, we have shown transfer of end-to-end controllers from simulation to the real world, where images and joint angles are continuously mapped directly to motor velocities, through a deep neural network. The capabilities of our method are demonstrated by learning a long-horizon multi-stage task that is analogous to a simple tidying task, and involves locating a cube, reaching the cube, grasping the cube, locating a basket, and finally dropping the cube into the basket. We expect the method to work well for other multi-stage tasks, such as tidying other rigid objects, stacking a dishwasher, and retrieving items from shelves, where we are less concerned with dexterous manipulation. However, we expect that the method as is, would not work in instances where tasks cannot be easily split into stages, or when the objects require more complex grasps. We also expect that longer tasks with many stages will require an increasingly larger dataset and network capacity. One key limitation of the work in this chapter is perhaps the decision to use imitation learning; although we are able to recover from small deviations from the dataset through our data augmentation, large deviations, such as dropping the cube or overshooting the basket, will not be recoverable. Perhaps more importantly, this method does not allow for continual learning, i.e. once the system is deployed, the method will not be able to learn from its mistakes. This motivated the use of an alternative learning method: reinforcement learning, which we investigate in the following chapter.

Chapter 4

Sim-to-Real Reinforcement Learning for Deformable Object Manipulation

Contents

4.1 Introduction

Much like our end-to-end method presented in Chapter 3, the majority of state-of-the-art work in robotic manipulation focuses on working with rigid objects, that either do not deform when they are grasped or have negligible deformation. However, deformable object manipulation has many important real-world applications. Key domains of interest are home assistance robotics (cloth folding [Miller et al., 2014], bed making [Laskey et al., 2017], getting dressed [Gao et al., 2016, Tamei et al., 2011]); medicine (robot surgery [Thananjeyan et al., 2017], suturing [Schulman et al., 2013]); and industry (cable insertion [Večerík et al., 2017]). Robots attempting to work with these objects are however presented with many new challenges, most notably the large object configuration spaces, the difficulty of accurate object behaviour modelling, and the large change in the configuration resulting from manipulation attempts.

Of the limited amount of work in deformable object manipulation, the majority focuses on folding 2D deformable objects, such as towels or articles of clothing. One approach employed explicit modelling of cloth deformation in simulation and then attempted to find an optimal trajectory based on the model [Li et al., 2015, Cusumano-Towner et al., 2011, Yamakawa et al., 2011]. However, those models tend to be very sensitive to the deformation parameters of the objects (stiffness, shear resistance, friction) and therefore do not generalise well to unseen objects or environments. The second approach does not attempt to model the cloth but instead relies on visuomotor servoing to achieve the task. The robot identifies ideal grasping points based on heuristics (e.g. large curvature corresponds to a corner) and then executes a folding routine [Maitin-Shepard et al., 2010, Osawa et al., 2007, Bersch et al., 2011]. Both approaches require a significant amount of engineering specific to the manipulation task, and it would be cumbersome to extend them to achieve success in a wholly different scenario. An alternative direction is to learn deformable object manipulation in an end-to-end manner, mapping observations directly to actions, and bypassing the need for explicit modelling. Specifically, we employ Reinforcement Learning (RL) to create an algorithm that is task agnostic and can learn many different behaviours based on the definition of a reward and a couple of provided demonstrations. This has been extensively studied in the context of rigid object manipulation (see [Quillen et al., 2018] for a comprehensive evaluation), but only a small amount of work has focused on deformable objects. Moreover, no study has previously investigated the applicability of sim-to-real methods (such as domain randomisation) to transfer deformable object policies. We believe that if sim-to-real methods are to be employed further, then it should be possible to learn to interact with a wide variety of objects, and not only rigid objects, which has been the case to-date. To the best of our knowledge, deep RL and sim-to-real have not yet been applied to the domain of deformable object manipulation.

In this chapter, we use an improved version of Deep Deterministic Policy Gradients (DDPG) [Lillicrap et al., 2015b], seeded with 20 demonstrations, to train an agent purely in simulation on three different tasks: folding a small towel diagonally, folding a towel up to a specific point and draping a towel over a small hanger. All tasks are learned via a single sparse reward on task completion. The agent receives only RGB images and the proprioceptive state (joint angles, gripper position) during test time. We employed domain randomisation [Tobin et al., 2017b, James et al., 2017b] in simulation to simplify the policy transfer from simulation to the real world without further training. Qualitative results can be seen in the video[1].

[1]https://sites.google.com/view/sim-to-real-deformable

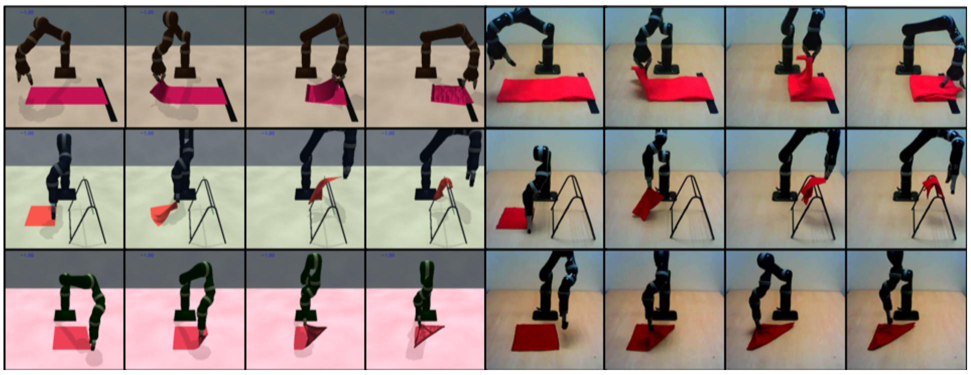


Figure 4.1: We learn robot policies in simulation and test them in the real-world. The algorithm was evaluated on 3 different tasks: folding a large towel up to a tape (top row), hanging a small towel on a hanger (middle row) and diagonally folding a square piece of cloth (bottom row).

Our contributions in this chapter are 2 fold. Firstly, we present the first work in transferring end-to-end controllers for deformable object manipulation through the use of domain randomisation. Secondly, we present and perform ablations on an improved version of Deep Deterministic Policy Gradients (DDPG) [Lillicrap et al., 2015b], which includes a number of extensions from the literature and brings considerable performance boosts.

4.2 Related Work

Cloth manipulation tasks solved by conventional robotics methods include cloth flattening [Sun et al., 2013], cloth folding [Yamakawa et al., 2011, Li et al., 2015] or bringing cloth into a desired configuration [Cusumano-Towner et al., 2011]. The robots identify the cloth configuration based on visual information with hand-engineered heuristics and then use this either directly to parametrise a pre-programmed trajectory or indirectly by feeding the information to a mathematical model of the cloth. Some methods have also leveraged demonstrations for cloth manipulation, either through the use of behavioural cloning with noise injection [Laskey et al., 2017, Lee et al., 2015b] or by creating a trajectory-aware registration method that becomes robust to distractions by observing the action multiple times [Lee et al., 2015b]. Other work has combined imitation learning and the PoWER RL algorithm to learn a policy for folding a towel by observing human demonstrations [Balaguer and Carpin, 2011]. The towel was equipped with reflective markers and a complex system was employed to reconstruct the missing data if the markers were occluded or not detected.

Reinforcement learning has not yet been extensively applied to cloth manipulation, even though it has found applications in many other robotic domains, including rigid object manipulation [Gu et al., 2016, Peters and Schaal, 2008], UAV control [Abbeel et al., 2006] or bipedal robot control [Peng et al., 2017]. One of the most prevalent deep RL methods in robotics is DDPG [Lillicrap et al., 2015b].

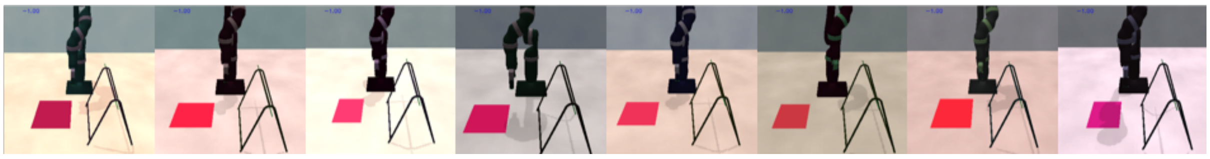


Figure 4.2: Examples of domain randomisation for the hanger environment. During randomisation, we vary the table textures, cloth and arm colours, light position, camera position and orientation, cloth size and position, hanger size and position, initial arm position and size of arm base.

The algorithm allows control in continuous space without discretisation, which makes it a good fit for controlling robot joint velocities. The algorithm has been the basis for a large number of extensions [Fujimoto et al., 2018b, Andrychowicz et al., 2017, Barth-Maro et al., 2018, Nair et al., 2017, Schaul et al., 2015b, Pinto et al., 2017] which have further improved the performance of the agent. DDPG can also be extended with demonstrations to considerably speed up the learning process [Večerík et al., 2017].

Transferring policies learned in simulation into the real world is a challenging task. Previous work has shown that direct transfer was not possible [James and Johns, 2016b], while others have shown that transfer only works after the agent has received additional training in the real world [Rusu et al., 2017b]. A promising technique to accomplish a successful transfer from simulation to the real world is domain randomisation [Tobin et al., 2017b, Pinto et al., 2017, James et al., 2017b], which samples simulation parameters (e.g. camera position, light position, textures etc.) from probability distributions centred at a noisy estimate of the ground truth. As a result, the agent learns to ignore minor variations in the environment, so it becomes robust to domain changes, including the sim-to-real transfer. This was the approach taken in the pick-and-place task, presented in Chapter 3.

4.3 Background

DDPG [Lillicrap et al., 2015b] is a deep RL algorithm for learning control policies in a continuous action domain. It uses an actor neural network, parametrised by a set of parameters θ^{π}, that maps observations to actions $\pi : O \rightarrow A$ and tries to maximise $Q(s_t, \pi(o_t))$ at each time-step t. However, the Q function is not known and DDPG employs a critic neural network, parametrised by parameters θ^{Q}, to estimate Q by minimising the Bellman loss:

$$L_{critic} = \left(Q(s_t, a_t) - r_t - Q'(s_{t+1}, \pi(o_{t+1}))\right)^2 .$$

During training, the agent acts in the environment according to noisy policy $a_t = \pi(o_t) + N(0, \sigma)$. The Gaussian noise facilitates exploration. Each transition the agent generates is stored in a replay buffer from where it is sampled in batches to train the networks. Sampling from a replay buffer stabilises training by removing temporal correlations and therefore reduces the changes in the distributions the networks are trying to learn. DDPG also employs a target networks Q' to reduce the risk of Q-value estimates oscillating or diverging due to the recursive Q-value definition in the Bellman equation.

DDPG became the primary building block of many other algorithms trying to improve on it. We give here a brief summary of the selected DDPG extensions that we incorporated into our algorithm.

Prioritised Replay Prioritised replay [Schaul et al., 2015b] assigns a priority p_i to each transition, computed as a sum of the last temporal difference (TD) error and small hyper-parameter ϵ. TD error is defined as the difference between critic prediction and critic target, so it serves as a proxy for the learning progress induced by the transition. ϵ guarantees that even transitions with small TD errors can be sampled in the future, which is necessary because the critic changes its estimates as learning progresses. All new transitions are added to the replay buffer with priority equal to the current maximal priority in the buffer. The sampling probability is computed as $P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$, where α is a parameter controlling the strength of the prioritisation. Prioritised sampling introduces a bias that needs to be corrected by multiplying the TD error of the transition when training by the importance sampling weight: $w_i = (\frac{1}{NP(i)})^\beta$, where β is a hyper-parameter controlling the magnitude of bias correction and N is the replay buffer size.

N-Step returns N-Step returns help to quickly propagate the reward signal throughout the robot trajectory by looking at N subsequent transitions instead of just one. It has been shown to accelerate and stabilise learning [Barth-Maro et al., 2018]. N-step returns change the critic loss to:

$$L_{nstep} = (Q(s_t, a_t) - \sum_{i=0}^{N} \gamma^i r_{t+i} - \gamma^N Q'(s_{t+N}, \pi(o_{t+N})))^2 \ .$$

It is possible to use both 1-step loss and N-step loss at the same time, in which case the critic loss becomes the sum of the losses weighted by two hyper-parameters λ_{nstep} and λ_{1step}.

DDPGfD The original DDPG usually does not perform well on complex multi-step tasks with sparse rewards, because it is statistically improbable that the agent would often discover the right behaviour by random exploration. DDPGfD [Večerík et al., 2017] overcomes this limitation by seeding the training with demonstrations, which are inserted into the prioritised replay buffer along with normal transitions. Demonstration transitions are never deleted from the replay buffer, and their priority is increased by a small constant ϵ_D to make them more likely to be sampled. DDPGfD begins with a **pre-training phase**, where it executes a fixed number of training steps using the replay buffer initialised with demo transitions. Following pre-training, it begins collecting new experiences. DDPGfD also employs N-Step returns and adds L2-regularisation on both actor and critic.

Behavioural Cloning [Nair et al., 2017] DDPG can be further adapted to take advantage of demonstrations by introducing behavioural cloning loss to the actor network. This loss is applied only when a demonstration is sampled from the replay buffer for training. It encourages the actor to propose the same action as the demonstrator in the given state. After sufficient training, the agent might surpass the performance of the demonstrator and L_{BC} would then become detrimental to agent performance. The Q-filter mitigates this problem by only applying L_{BC} if the critic judges that the action proposed by the actor is worse than the action of the demonstrator.

$$L_{BC} = \begin{cases} |\pi(o_i) - a_i|^2, & \text{if } Q(s_i, a_i) > Q(s_i, \pi(o_i)) \\ 0 & \text{otherwise} \end{cases}$$

Reset to demonstration Reset to demonstration [Nair et al., 2017] aims to make it easier for the agent to receive a reward in sparse long-horizon tasks. After the end of an episode, the environment will have a small probability of being placed into a random state encountered during demonstrations. In those cases, the agent only needs to complete the sub-task starting at the sampled state. This sub-task is usually substantially easier, particularly if the demonstration state was sampled near the end of the episode.

TD3 DDPG is prone to overestimating Q-values, which in turn leads to sub-optimal policies. TD3 [Fujimoto et al., 2018b] implements 3 improvements to address the overestimation resulting from approximation errors. Firstly, it maintains 2 independent critic networks and always takes the minimum Q-value as the optimisation target for both actor and critic. Secondly, it proposes to delay the propagation of weight updates to target network by a couple of steps, so they have time to converge to a better quality update. Finally, it regularises the target Q-value by adding a clipped normal noise to the action proposed by the target actor to explicitly increase the smoothness of the Q-function prediction. The TD3 1-step target of the critic is defined to be:

$$y = r_t + \min_{i=1,2} Q'_i(s_{t+1}, \pi(o_{t+1}) + \text{clip}(\mathcal{N}(0,\sigma), -c, c)) \, .$$

Asymmetric actor-critic The simulator always has a perfect understanding of the environment, which can be leveraged during the training phase. Asymmetric actor-critic [Pinto et al., 2017] uses high dimensional (RGB) partial observations as an input to the actor, whilst using low-dimensional environment state (object positions, arm state, etc.) as the input for the critic. This extension significantly reduces the number of trainable parameters and increases the accuracy of the critic.

4.4 Method

4.4.1 Simulation

This chapter is the only one that does not use CoppeliaSim for the simulation environment, and instead uses Pybullet [Coumans and Bai, 2016]. This is because CoppeliaSim does not have support for deformable objects. Pybullet implements some rudimentary and experimental functionality for simulating deformable objects. Even though the simulator implements 2D rectangular cloth creation in its C++ API, we found the out-of-the-box simulation behaviour impractical for our purposes. We initially tried to rely on physics simulation to create a lasting grasp, which was not possible. The gripper either tunnelled through the cloth (low collision margin) or the gripper repelled it before the grasp attempt (high collision margin). We were only able to resolve the issue by creating a fake grasp implemented as a set of anchors between cloth nodes and gripper fingers.

The grasp creation was stochastic and deliberately failed in 5% of the cases to expose the agent to unsuccessful grasp scenarios. Moreover, the creation of the constraint was subject to the gripper endpoint being in close proximity to a cloth node. Creating the constraint only to a single point on each gripper causes the cloth to spin unnaturally, so multiple anchors were used — one at the middle and one at both extremities of each fingertip. Finally, we found that the existing implementation of anchors between soft bodies and rigid bodies was not sufficient because it reached an equilibrium of forces with the cloth hanging approximately 5 cm below the gripper. We adapted the implementation so the anchor between cloth node and rigid object is honoured regardless of other forces acting on the cloth.

We employed domain randomisation to facilitate a smooth domain transfer of the learned policy. More specifically, we randomised the textures using Perlin noise [Perlin, 1985]; object and background colours; object parameters and positions; arm spawn position and joint angles; camera position, orientation and intrinsics; light source position and colour; and all reflectance coefficients. The values were sampled from either normal or uniform distributions around the noisy ground truth estimates.

4.4.2 Learning algorithm with integrated improvements

During initial experimentation, we found that DDPG was not successful in solving any of the proposed environments, and so investigated possible improvements. We have taken inspiration from the success of the Rainbow DQN agent [Hessel et al., 2017] integrating all recently proposed extensions and achieving state-of-the-art performance on a set of benchmark tasks. Starting with the DDPG baseline available in the OpenAI repository [Dhariwal et al., 2017], we implemented all DDPG extensions listed in Section 4.3. We however did not use the Q-value target regularisation in TD3 because we found it to be

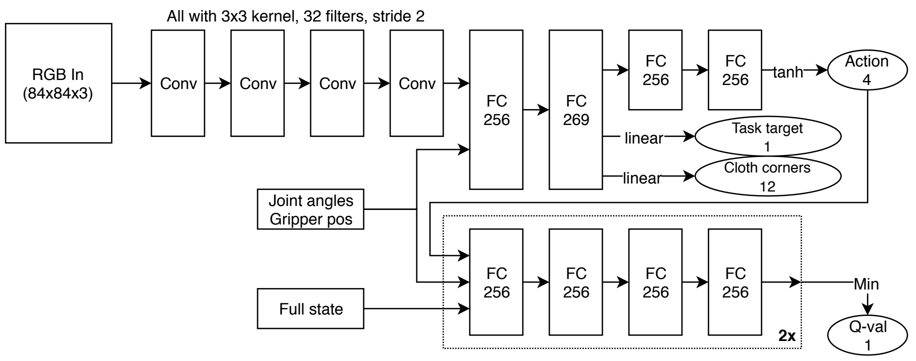


Figure 4.3: The network architecture uses 3 different inputs — RGB images from the camera looking at the scene, joint angles and gripper position (available at test time from the robot API) and full state, which is only available at training time. The top half of the figure corresponds to the actor, while the bottom half corresponds to twin critics. The actor receives joint angles, gripper position and RGB images while the critic receives full-low dimensional state. Auxiliary outputs of the actor are only used during training to help the network quickly recognise essential scene features. However, they were also useful for debugging purposes at test time, because we can plot the estimate of cloth position and target position to verify that the actor understands the scene.

detrimental to the agent performance for these particular tasks. This results in the following critic loss, applied to a_t during training:

$$
\begin{aligned}
L_{critic}(a) &= \lambda_{nstep} L_{nstep}(a) w_i + \lambda_{1step} L_{1step}(a) w_i + \lambda_{L2} L^{Q}_{reg}(\theta^{Q}), \\
L_{nstep}(a) &= \Big(Q(s_t, a) - \sum_{i=0}^{N} \gamma^{i} r_{t+i} - \gamma^{N} \min_{i=1,2} Q'_i(s_{t+N}, \pi(o_{t+N}))\Big)^2, \\
L_{1step}(a) &= \Big(Q(s_t, a) - r_t - \min_{i=1,2} Q'_i(s_{t+1}, \pi(o_{t+1}))\Big)^2.
\end{aligned}
$$

The auxiliary outputs predict the key features of the environments (in our case those are cloth corner positions, tape y-coordinate and hanger y-coordinate). L_{aux} is the mean square error between the prediction and the actual value. Each component of the auxiliary predictions can be weighted by separate weightings, although this was rarely used in practice. The resulting actor loss is:

$$
L_{actor} = -L_{critic}(\pi(o_t)) + \lambda_{BC} L_{BC} + L_{aux}
$$

$$
L_{BC} = \begin{cases} (\pi(o_i) - a_i)^2, & \text{if } Q(s_i, a_i) > Q(s_i, \pi(o_i)) \text{ and } i \text{ is demonstration} \\ 0 & \text{otherwise .} \end{cases}
$$

The priority of each transaction is updated after each training step according to:

$$
p_i = L_{i,nstep}(a_i) + L_{i,1step}(a_i) + \epsilon + \epsilon_D \max_{k \in minibatch} \left(L_{k,nstep}(a_k) + L_{k,1step}(a_k) \right),
$$

where $\epsilon = 10^{-6}$ is a small constant. We found that it was impossible to tune the fixed constant ϵ_D (as suggested by DDPGfD) to boost the priority of demonstrations further because the TD error magnitude varied by multiple orders of magnitude across training epochs. We instead made the further demo priority boost term proportional to the maximal losses in the current mini-batch. ϵ_D is set to 0 for updating priorities of all transitions apart from demonstrations. We used the same network architecture (Figure 4.3) for all 3 experiments. The full learning algorithm with all improvements are available online.[2].

4.5 Experiments

4.5.1 Cloth manipulation environments

All standard RL environments for manipulation tasks only contain rigid objects, so we designed and implemented 3 new environments for solving deformable object tasks. Each environment exposes an RGB observation with dimensions 84x84x3, a low dimensional state and low dimensional actor input (joint angles and gripper position). The robotic arm in the environments is 7DOF Kinova Mico controlled by 4-dimensional action. First 3 dimensions are the velocity of the end effector while the last dimension is a gripping velocity (negative for opening and positive for closing). The reward is sparse with +100 for success and 0 otherwise. Gripper rotation is not necessary for the tasks and is therefore kept fixed. The origin of the coordinate system is at the base of the arm, with z-axis perpendicular to the table and x-axis pointing towards the camera. The environments implement OpenAI gym [Brockman et al., 2016a] API and use Pybullet as a simulation engine [Coumans and Bai, 2016]. We call the 3 environments Tape, Hanging and Diagonal Folding:

1. **Tape**: The robot needs to fold a large towel up to a mark identified by a piece of black tape. The tape can be in 3 different positions: 5/8th, 7/8th and at the end of the towel. The robot receives a reward if both corners of the lifted side of the cloth are within a threshold distance from the tape. The gripper is fixed to point downwards with fingers parallel to the y-axis. This task was proposed by [Lee et al., 2015b].
2. **Hanging**: The robot needs to grasp the piece of cloth and drape it over a small hanger. The cloth appears on the left side of the scene, and we sample its position from a uniform distribution. The reward is given when the cloth is released from the gripper, and all corners stay 5 or more cm over the ground for 20 simulation steps (this rules out cloth sliding off the hanger). The gripper has fingers parallel to the x-axis.
3. **Diagonal folding**: The robot needs to fold a rectangular face towel ($\sim 28 \times 28$cm) diagonally. The reward is given if the diagonal corners are within a threshold distance from each other and all pairs of corners on the same side of the rectangle are at distances larger than 3/4 of the side length

[2]https://sites.google.com/view/sim-to-real-deformable

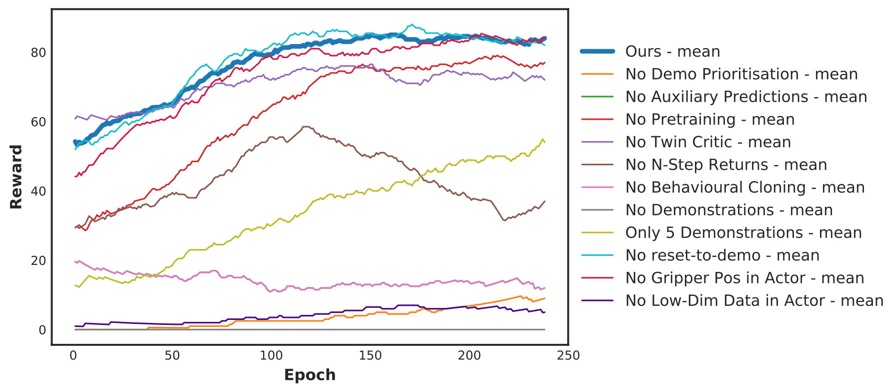


Figure 4.4: Ablation studies on the Diagonal Folding task, where "Ours" shows the result of the algorithm with all improvements. The reward for success was set to be 100, and therefore it is equal to the percentage of successes. Two evaluation episodes were performed after each epoch. Curves report the mean of 2 random seeds, and they were smoothed to improve legibility.

when flat (this is to prevent the robot simply crumpling the cloth to align corners, which we have observed before). The gripper is parallel to the x-axis.

4.5.2 Simulation results and ablation studies

We ran the training algorithm with all implemented improvements (labelled "Ours") on the three task, as none of the improvements had an adverse effect. Each training run was seeded with 20 demos. Each experiment took approximately 24 hours to run on one GeForce GTX TITAN. The success rates (mean of 3 random seeds) in the final evaluations of the experiments are shown in Table 4.1, which were achieved after approximately 80k transitions and in the presence of domain randomisation.

Success rates (Sim)	
Diagonal folding	90%
Hanging	77%
Tape	86%

Table 4.1: Success rates in simulation

The most likely failure case across all environments is a failure to grasp the cloth. Even though the agent has learned to do multiple re-grasps, in some situations, it repeatedly fails (e.g. by closing the gripper above the towel). We believe this is due to an outlier in the camera configuration sampled from a normal distribution. Secondly, too fast or inaccurate motion usually causes the agent to crumple the towel after which it is no longer able to achieve the task. Thirdly, in the Hanging task, the agent sometimes drapes the cloth too far, causing it to fall.

We performed ablation studies to verify the contributions of selected modifications to DDPG. The agent integrating all improvements either outperforms or matches the performance of all training runs with an ablation. Two implemented improvements do not seem to increase the agent performance: reset to demonstration and adding gripper position to the low dimensional actor input. In the first case, we

Hanging task	
Vicinity	100%
Grasp	76.6%
Drape over	70%
Full success	46.6%

Diagonal folding task	
Grasp	66.6%
Not crumpled	66.6%
$d \leq 0.15$m	53.3%
$d \leq 0.1$m	40%
$d \leq 0.05$m	20%

Tape folding task	
Grasp	90%
$d \leq 0.15$m	90%
$d \leq 0.1$m	76.6%
$d \leq 0.05$m	43%

Table 4.2: The success rates for each environment in the real world. Note that these are run in the real world without additional training. For the hanging task, *vicinity* means the gripper being within 5cm from the cloth, *drape over* means the cloth is touching the top part of the hanger and *full success* is achieved if the cloth does not fall after it is released. For diagonal folding, *not crumpled* means that adjacent corners are more than 15cm from each other and the d is the distance between diagonal corners (lower is better). For tape folding, d is the distance between towel edge and the tape mark.

hypothesise that due to the BC loss, the agent can complete successful full-length tasks early in training so it can quickly form a diverse set of successful episodes. This might be preferable over repeatedly resetting to similar states from demonstrations. In the second case, we removed gripper position from actor input, instead making it an auxiliary output. The agent accurately learned to predict the forward kinematics, so the gripper position input was not necessary. However, also removing joint angles (No Low-Dim Data in actor) was detrimental to performance which indicates that the agent cannot infer gripper position accurately from images only.

The features with questionable value are Twin Critic and Pre-training. Although they seem to provide improvement, the trade-off is increased computational cost. Pre-training has a constant cost of 7 minutes at the start of training and maintaining two critics increased runtime by 1%. However, Twin Critic would be substantially more expensive if it also used RGB observations.

The improvements that convincingly demonstrated a positive contribution to agent performance are Auxiliary predictions, Behavioural Cloning and Demo prioritisation. Without boosting the priority of the demonstrations (adding the ϵ_D term to priority equation), they are much less likely to be sampled because they form only a tiny portion of the replay buffer.

4.5.3 Sim-to-real transfer

In real-world experiments, we use the Kinova Mico 7DOF robotic arm mounted in the middle of a table, and we collect the RGB observation using a low-cost Genius C170 web camera mounted on a fixed tripod next to the table. We report the results of 30 trials on the real robot for each task in Table 4.2. As in simulation, the most prominent failure case is failed grasping, particularly with thin face towels (used in Hanging and Diagonal folding). The robot has only a small acceptable margin of error (roughly 1 cm) in the z-axis for a successful grasp — going too low will prevent the gripper from closing and going too high will not grasp the cloth. The other common failure case was an imprecise movement resulting in crumpling of the fabric from which the agent was not able to recover. this was partly caused by low simulation fidelity. The real cloth was much stiffer and therefore less forgiving to imprecise movement and the agent could not learn this in simulation.

When experimenting with various levels of domain randomisation, we found that heavy randomisation can be detrimental to learning. Specifically, we tried sampling the texture colours from a uniform distribution across all colours and the performance of the agent after the transfer was significantly worse. We believe that it then became much harder for the network to identify invariant environment features it could use for orientation. Consistently with the work presented in Chapter 3, we found that camera randomisation is essential for successful transfer; even with randomisation, the agent was still very sensitive to the camera position.

4.6 Conclusion

Building up on recent work in end-to-end learning for rigid object manipulation, we have extended those ideas to the domain of deformable objects and specifically, we have addressed the problem of cloth manipulation. We proposed a task agnostic algorithm based on Deep RL which bypasses the need to explicitly model cloth behaviour and does not require reward shaping to converge. The agent was able to learn 3 long horizon tasks: folding a towel to a tape mark, diagonal folding of face towel and draping a small towel over a hanger. Training was seeded with 20 demonstrations and happened entirely in simulation with a couple of adaptations to account for imperfections in experimental deformable body support, and with domain randomisation to enable easy transfer of the policy. The learning algorithm incorporated 9 improvements proposed in the recent literature and we have presented ablation studies to understand the role of these improvements.

The core limitations of this work lie in the simulation itself. Despite making various changes to the Pybullet codebase to improve deformable simulation, many issues still remained. One of the biggest issues was a lack of self-collision capability which resulted in a number of visual and behavioural artefacts. Another issue was the lack of deformation stability. In the real world, when a cloth is crumpled, it tends to stay in a crumpled state. However, in our simulation, the cloth would slowly unroll into its original shape. To some degree this can be mitigated by selecting correct cloth parameters (large mass, small stiffness coefficients, large damping), but a change to these values can then negatively affect the behaviour of the cloth in other situations. Because of this, many tasks that would be appealing to learn in simulation, such as cloth dewrinkling or flattening, were not able to be correctly simulated.

Despite these issues, the goal of this work was not to create an accurate cloth environment, but to explore transferring deformable manipulation policies from simulation to reality. The fact that we were able to achieve success despite all of the simulation issues is encouraging, and suggest that there are easy real-world performance gains to be had by simply improving simulation fidelity. One of our real-world failure modes supports this claim. In the diagonal folding task, rather than moving straight from one corner to another, the agent would often lift the cloth too high and perform a suboptimal trajectory which deviated to a side, resulting in the cloth crumpling. By investigating the trained agent in simulation, we found that the same motion was performed, but the cloth behaved differently and was able to achieve the reward. This was due to its lower linear stiffness that would allow it to stretch when pulled up, and avoid crumpling.

To conclude this chapter, we believe that the primary factor limiting further research into deformable object manipulation is the lack of support for those objects in most robotic simulators. We are hoping that further research into simulation will allow us to create an accurate model of deformable object grasping, incorporate it into a widely used simulator and release the environments to create a set of benchmark tasks for future research in the domain.

CHAPTER 5

Task-Embedded Control Networks

Contents

5.1 Introduction

Humans and animals are capable of learning new information rapidly from very few examples, and apparently improve their ability to 'learn how to learn' throughout their lives [Harlow, 1949]. Endowing robots with a similar ability would allow for a large range of skills to be acquired efficiently, and for existing knowledge to be adapted to new environments and tasks. An emerging trend in robotics is to learn control directly from raw sensor data in an end-to-end manner. Such approaches have the potential to be general enough to learn a wide range of tasks, and they have been shown to be capable

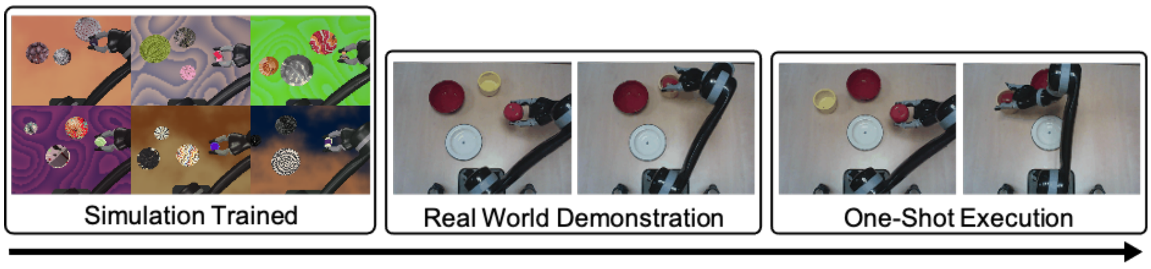


Figure 5.1: The robot gains its few-shot learning ability in simulation, and can then learn a new task from a single demonstration.

of performing tasks that traditional methods in robotics have found difficult, such as when close and complicated coordination is required between vision and control [Levine et al., 2016a], or in tasks with dynamic environments [James et al., 2017a]. However, these solutions often learn their skills from scratch and need a large amount of training data [James et al., 2017a, Zhang et al., 2018, James and Johns, 2016a]. A significant goal in the community is to develop methods that can reuse past experiences in order to improve the data efficiency of these methods.

To that end, one significant approach is Meta-Imitation Learning (MIL) [Finn et al., 2017b], in which a policy is learned that can be quickly adapted, via one or few gradient steps at test time, in order to solve a new task given one or more demonstrations. The underlying algorithm, Model-Agnostic Meta-Learning (MAML) [Finn et al., 2017a] can be very general, but lacks some of the properties that we might hope for in a robotic system. For one, once the policy is trained, it cannot accomplish any of the tasks seen during training unless it is given an example again at test time. Also, once a specific task is learned, the method can lose its ability to meta-learn and be stuck with a set of weights that can only be used for that one task. One way around this is to make a copy of the weights needed for each task, but this raises scalability concerns.

Our new approach, Task-Embedded Control Networks (TecNets), is centred around the idea that there is an embedding of tasks, where tasks that are similar (in terms of visual appearance) should be close together in space, whilst ones that are different should be far away from one another. Having such an expressive space would not only allow for few-shot learning, but also opens the possibility of inferring information from new and unfamiliar tasks in a zero-shot fashion, such as how similar a new task may be to a previously seen one.

TecNets, which are summarised in Figure 5.2, are composed of a *task-embedding network* and a *control network* that are jointly trained to output actions (e.g. motor velocities) for a new variation of an unseen task, given a single or multiple demonstrations. The task-embedding network has the responsibility of learning a compact representation of a task, which we call a *sentence*. The control

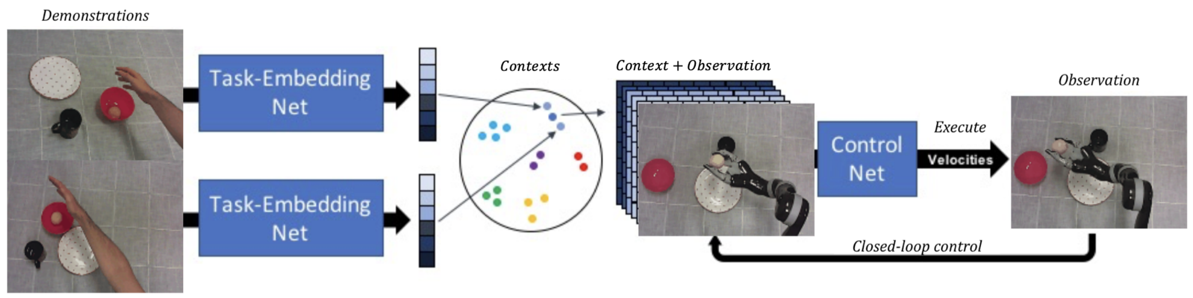


Figure 5.2: Task-Embedded Control Networks (TecNets) allow tasks to be learned from single or multiple demonstrations. Images of demonstrations are embedded into a compact representation of a task, which can be combined to create a *sentence*. This sentence is then expanded and concatenated (channel-wise) to the most recent observation from a new configuration of that task before being sent through the control network in a closed-loop manner. Both the task-embedding net and control net are jointly optimised to produce a rich embedding.

network then takes this (static) sentence along with current observations of the world to output actions. TecNets do not have a strict restriction on the number of tasks that can be learned, and do not easily forget previously learned tasks during training, or after. The setup only expects the observations (e.g. visual) from the demonstrator during test time, which makes it very applicable for learning from human demonstrations.

To evaluate our approach, we present simulation results from two experimental domains proposed in MIL [Finn et al., 2017b], and demonstrate that we can train our meta-learning ability in simulation and then deploy in the real world without any additional training. We believe this to be a desirable property given that large amounts of data are needed to end-to-end solutions. Despite being trained to meta-learn in simulation, the robot can learn new tasks from a single demonstration in the real world.

Our contributions in this work are threefold. We demonstrate the ability to one-shot and few-shot learn visuomotor control through the use of TecNets in a selection of visually-guided manipulation tasks. Secondly, we show that TecNets are able to achieve higher success rates compared to MIL [Finn et al., 2017b] when using only visual information from each demonstration. Finally, we demonstrate the first successful method of a few-shot learning approach trained in simulation and transferred to the real world, which we believe is an important direction for allowing large-scale generalisation.

5.2 Related Work

Our work lies at the intersection of imitation learning [Schaal, 1999, Argall et al., 2009] and meta-learning [Thrun and Pratt, 2012, Lemke et al., 2015]. Imitation learning aims to learn tasks by observing a demonstrator. One focus within imitation learning is *behavioural cloning*, in which the agent learns a mapping from observations to actions given demonstrations, in a supervised learning manner [Pomerleau, 1989, Ross et al., 2011]. Another focus is *inverse reinforcement learning* [Ng et al., 2000], where an agent attempts to estimate a reward function that describes the given demonstrations [Abbeel and Ng, 2004, Finn et al., 2016]. In our work, we focus on behavioural cloning in the context of learning motor control directly from pixels. A common issue in behavioural cloning is the large amount of data needed to train such systems [James et al., 2017a], as well as the fact that tasks are often learned independently, where learning one task does not accelerate the learning of another. Recently, there has been encouraging work to address this problem [Finn et al., 2017b], and our approach provides a further advance.

One-shot and few-shot learning is the paradigm of learning from a small number of examples at test time, and has been widely studied in the image recognition community [Vinyals et al., 2016, Koch et al., 2015, Santoro et al., 2016, Ravi and Larochelle, 2017, Triantafillou et al., 2017, Snell et al., 2017]. Many one-shot and few-shot learning methods in image recognition are a form of meta-learning, where the algorithms are tested on their ability to learn new tasks, rather than the usual machine learning paradigm of training on a single task and testing on held out examples of that task. Common forms of meta-learning include recurrence [Santoro et al., 2016], learning an optimiser [Ravi and Larochelle, 2017], and more recently Model Agnostic Meta-Learning (MAML) [Finn et al., 2017a]. Many works in metric learning, including ours, can be seen as forms of meta-learning [Vinyals et al., 2016, Snell et al., 2017], in the sense that they produce embeddings dynamically from new examples during test time; the difference to other more common meta-learning approaches is that the embedding generation is fixed after training.

The success of our new approach comes from learning a metric space, and there has been an abundance of work in metric learning for image classification [Kulis et al., 2012, Bellet et al., 2013], from which we will summarise the most relevant. Matching Networks [Vinyals et al., 2016] use an attention mechanism over a learned embedding space which produces a weighted nearest neighbour classifier given labelled examples (support set) and unlabelled examples (query set). Prototypical Networks [Snell et al., 2017] are similar, but differ in that they represent each class by the mean of its examples (the prototype) and use a squared Euclidean distance rather than the cosine distance. In the case of one-shot learning, matching networks and prototypical networks become equivalent. Our approach is similar in that our sentence (prototype) is created by averaging over the support set, but differs in the way we couple the learned embedding space with the control network. These metric learning methods have all been developed for image classification, and in the visuomotor control domain of our method, we do not explicitly classify sentences, but instead jointly optimise them with a control network.

Recently, [Hausman et al., 2018] proposed learning a skill embedding space via reinforcement learning that led to speed-ups during training time. Although impressive, that method does not focus on few-shot learning, and the experiments are run within simulation with low dimensional state spaces. Another piece of work that uses embedding spaces is [Sung et al., 2017], where a multimodal embedding is learned for point-clouds, language and trajectories. This work involves pre-training the parts of the network, and also relies on accurate models of the world. Our approach has the benefit that we map directly from images to motor actions and train jointly embedding and control networks, with no pre-training.

In terms of setup, the closest related work to ours is MIL [Finn et al., 2017b], where they apply MAML [Finn et al., 2017a] and behaviour cloning to learn new tasks, end-to-end from one visual demonstration. The underlying algorithm, MAML, learns a set of weights that can be quickly adapted to new tasks. If we were to use this approach to retain information we had previously learnt, we would need to hold copies of weights for each task. In comparison, our method relies on storing a compact sentence for every task we want to remember.

5.3 Task-Embedded Control Networks

We now formally summarise the notation for our method. A policy π for task $\mathfrak{T}$ maps observations $\mathbf{o}$ to actions $\boldsymbol{a}$, and we assume to have expert policies π^* for multiple different tasks. Corresponding example trajectories consist of a series of observations and actions: $\tau = [(\mathbf{o}_1, \boldsymbol{a}_1), \dots, (\mathbf{o}_T, \boldsymbol{a}_T)]$ and we define each task to be a set of such examples, $\mathfrak{T} = \{\tau_1, \cdots, \tau_K\}$. TecNets aim to learn a universal policy $\pi(\mathbf{o}, \boldsymbol{s})$ that can be modulated by a sentence $\boldsymbol{s}$, where $\boldsymbol{s}$ is a learned description of a task $\mathfrak{T}$. The resulting universal policy $\pi(\mathbf{o}, \boldsymbol{s})$ should emulate the expert policy π^* for task $\mathfrak{T}$.

5.3.1 Task Embedding

We now introduce our task embedding, which can be used independently in other fields, such as image classification, and so we keep this section general. Assume we are given a small set of K examples of a task $\mathfrak{T}^j$. Our task embedding network $f_\theta : \mathbb{R}^D \rightarrow \mathbb{R}^N$ computes a normalised N-dimensional vector $\boldsymbol{s}_k^j \in \mathbb{R}^N$ for each example $\tau_k^j \in \mathfrak{T}^j$. A combined sentence $\boldsymbol{s}^j \in \mathbb{R}^N$ is then computed for that task by taking the normalised mean of the example vectors:

$$\boldsymbol{s}^j = \left[\frac{1}{K} \sum_{\tau_k^j \in \mathfrak{T}^j} f_\theta(\tau_k^j) \right]^\wedge , \tag{5.1}$$

where $\boldsymbol{v}^\wedge = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$. We then need to define a loss function that can be used to learn an ideal embedding. We use a combination of the cosine distance between points and the hinge rank loss (inspired by [Frome

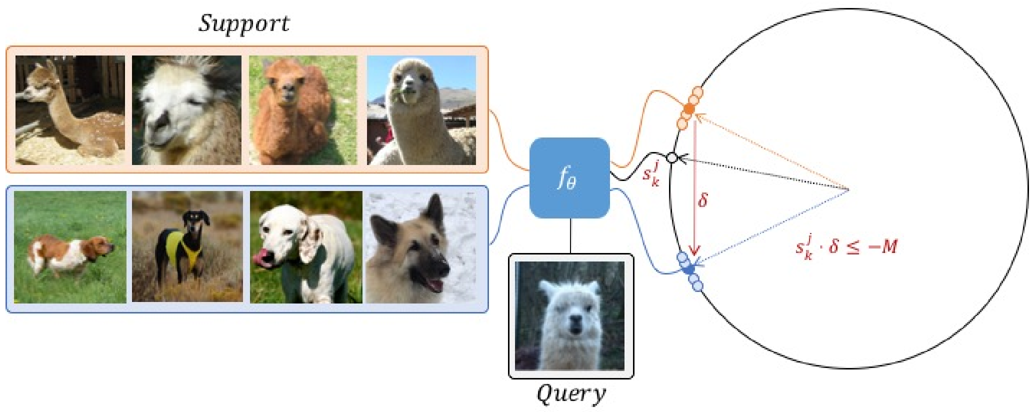


Figure 5.3: A visualisation of how the embedding is learned. Imagine a simple case where we have 2 tasks (or classes): llamas and dogs. We have a support set of 4 examples, which are then embedded and averaged in order to get a sentence for each task. The hinge rank loss drives the dot product of the query image ($\boldsymbol{s}_k^j$) with the difference between the actual sentence and the negative sentences (δ) to be at least a factor of margin away (M in the Figure above). In other words, $\boldsymbol{s}_k^j$ should point in the opposite direction to δ by at least a factor of margin.

et al., 2013]). The loss for a task $\mathfrak{T}^j$ is defined as:

$$\mathcal{L}_{emb} = \sum_{\mathcal{T}_k^j \in \mathfrak{T}^j} \sum_{\mathfrak{T}^i \neq \mathfrak{T}^j} max[0, margin - \boldsymbol{s}_k^j \cdot \boldsymbol{s}^j + \boldsymbol{s}_k^j \cdot \boldsymbol{s}^i] \,, \tag{5.2}$$

which trains the model to produce a higher dot-product similarity between a task's example vectors $\boldsymbol{s}_k^j$ and its sentence $\boldsymbol{s}^j$ than to sentences from other tasks $\boldsymbol{s}^i$. We illustrate the intuition behind this loss in Figure 5.3.

Additionally, we pick two disjoint sets of examples for every task $\mathfrak{T}^j$: a support set $\mathfrak{T}_U^j$ and a query set $\mathfrak{T}_Q^j$. In the above embedding loss, the support set is used to compute the task sentences, $\boldsymbol{s}^i$ and $\boldsymbol{s}^j$, and only the examples picked from the query set are used as example vectors, $\boldsymbol{s}_k^j$. Given that each of the sampled tasks in a training batch are unique, the negatives $\mathfrak{T}^i$ can be chosen to be all the other tasks in the batch. Therefore, for each task within a batch, we also compare to every other task. Further details are given in Algorithm 1.

In all of our experiments, we set $margin = 0.1$, though in practice we found a wide range of values between $0.01 \leq margin \leq 1.0$ that would work. Although not used, we can treat this embedding as a classification of tasks, whose accuracy we can estimate by computing the sentence $\boldsymbol{s}_k$ of an example and then performing a nearest neighbour search in the embedding space over all task sentences $\boldsymbol{s}^j$. In addition to the dot-product similarity and hinge rank loss, we also tried other distances and losses. One such distance and loss was the squared Euclidean distance used in [Snell et al., 2017], but we found that this did not work as well for our case.

Algorithm 1 Training loss computation for one batch. B is the batch size, K_U and K_Q are the number of examples from the support and query set respectively, and $RandomSample(S, N)$ selects N elements uniformly at random from the set S.

```
 1: procedure TRAINING ITERATION
 2:     $\mathcal{B} = RandomSample(\{\mathfrak{T}_1, \cdots, \mathfrak{T}_N\}, \text{B})$
 3:     for $\mathfrak{T}^j \in \mathcal{B}$ do
 4:         $\mathfrak{T}^j_U = RandomSample(\mathfrak{T}^j, K_U)$
 5:         $\mathfrak{T}^j_Q = RandomSample(\mathfrak{T}^j \backslash \mathfrak{T}^j_U, K_Q)$
 6:         $\boldsymbol{s}^j_U = \left[\frac{1}{K_U} \sum_{\tau \in \mathfrak{T}^j_U} f_\theta(\tau)\right]^\wedge$
 7:         $\boldsymbol{s}^j_q = f_\theta(\tau_q) \quad \forall \tau_q \in \mathfrak{T}^j_Q$
        $\mathcal{L}_{emb} = \mathcal{L}^U_{ctr} = \mathcal{L}^Q_{ctr} = 0$
 8:     for $\mathfrak{T}^j \in \mathcal{B}$ do
 9:         $\mathcal{L}_{emb} \mathrel{+}= \sum_q \sum_{i \neq j} max[0, margin - \boldsymbol{s}^j_q \cdot \boldsymbol{s}^j_U + \boldsymbol{s}^j_q \cdot \boldsymbol{s}^i_U]$
10:         $\mathcal{L}^U_{ctr} \mathrel{+}= \sum_{\tau \in \mathfrak{T}^j_U} \sum_{(\mathbf{o}, \boldsymbol{a}) \in \tau} \|\pi(\mathbf{o}, \boldsymbol{s}^j_U) - \boldsymbol{a}\|^2_2$
11:         $\mathcal{L}^Q_{ctr} \mathrel{+}= \sum_{\tau \in \mathfrak{T}^j_Q} \sum_{(\mathbf{o}, \boldsymbol{a}) \in \tau} \|\pi(\mathbf{o}, \boldsymbol{s}^j_U) - \boldsymbol{a}\|^2_2$
12:     $\mathcal{L}_{tec} = \lambda_{emb} \mathcal{L}_{emb} + \lambda^U_{ctr} \mathcal{L}^U_{ctr} + \lambda^Q_{ctr} \mathcal{L}^Q_{ctr}$
13:     return $\mathcal{L}_{tec}$
```

Algorithm 2 How TecNets operate during test time. D is the set of demonstrations for a task, Env is the environment in which to act.

```
1: procedure TEST($D, Env$)
2:     $\boldsymbol{s} = \left[\frac{1}{|D|} \sum_{\tau \in D} f_\theta(\tau)\right]^\wedge$
3:     while task not complete do
4:         $\mathbf{o} = Env.GetObservation()$
5:         $\boldsymbol{a} = \pi(\mathbf{o}, \boldsymbol{s})$
6:         $Env.Act(\boldsymbol{a})$
```

5.3.2 Control

In contrast to metric learning systems for classification, which would use some sort of nearest neighbour test to find the matching class, here the embedding is relayed to the control network and both networks are trained jointly. Given a sentence $\boldsymbol{s}^j_U$, computed from the support set $\mathfrak{T}^j_U$, as well as examples from the query set $\mathfrak{T}^j_Q$ we can compute the following loss for the policy π:

$$\mathcal{L}^Q_{ctr} = \sum_{\tau^j_q \in \mathfrak{T}^j_Q} \sum_{(\mathbf{o}, \boldsymbol{a}) \in \tau^j_q} \|\pi(\mathbf{o}, \boldsymbol{s}^j_U) - \boldsymbol{a}\|^2_2 \,. \tag{5.3}$$

This allows the embedding not only to be learned from the embedding loss $\mathcal{L}_{emb}$, but also from the control loss, which can lead to a more meaningful embedding for the control network than if they were trained independently. Though appropriate weightings must be selected, as the control network needs the embedding in order to know which task to perform, but the embedding network may have to wait for a reasonable control output before being able to enrich its structure.

We found it helpful for the control network to also predict the action for the examples in the support

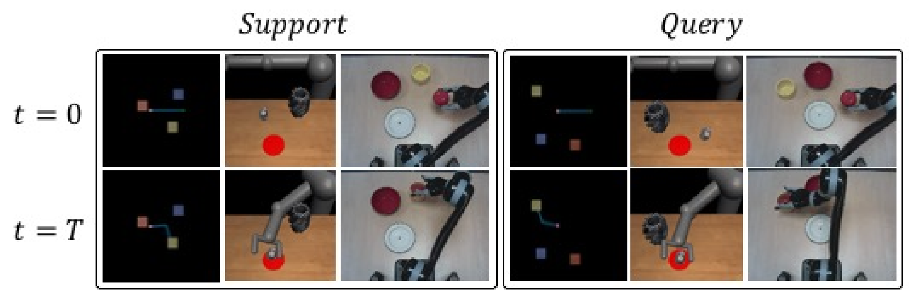


Figure 5.4: Here we show the first and last timestep of a single example (i.e. one-shot) from the support and query set for each of the 3 experimental domains. The support examples are used to describe the task, whilst the query set examples test the networks ability to perform a modified version of the task. We now highlight each of the tasks in the support set. *Left:* the simulated reaching experiment, where the robot must reach a specified colour. *Centre:* the simulated pushing experiment, where the robot must push a specified object to the red target. *Right:* the real world placing experiment, where the robot must place an item into a specified container.

set $\mathfrak{T}_U^j$ and training with a control loss $\mathcal{L}_{ctr}^U$. This has the advantage that it makes the task of learning $\mathcal{L}_{ctr}^Q$ easier, as learning $\mathcal{L}_{ctr}^U$ can be seen as an easier version of minimising the former (since example dependent information can be passed through the embedding space). Thus, the final loss is:

$$\mathcal{L}_{Tec} = \sum_{\mathfrak{T}} \lambda_{emb}\mathcal{L}_{emb} + \lambda_{ctr}^U \mathcal{L}_{ctr}^U + \lambda_{ctr}^Q \mathcal{L}_{ctr}^Q. \tag{5.4}$$

Input to the task-embedding network consists of $(width, height, 3 \times |\tau|)$, where 3 represents the RGB channels. For all of our experiments, we found that we only need to take the first and last frame of an example trajectory τ for computing the task embedding and so discarded intermediate frames, resulting in an input of $(width, height, 6)$. The sentence from the task-embedding network is then tiled and concatenated channel-wise to the input of the control network (as shown in Figure 5.2), resulting in an input image of $(width, height, 3\text{+N})$, where N represents the length of the vector. Pseudocode for both the training and testing is provided in Algorithms 1 and 2 respectively.

5.4 Experiments

In this section, we aim to answer the following: (1) Is it possible to learn a task embedding that can directly be used for visuomotor control? (2) Does our metric loss lead to a better embedding rather than allowing the control network to have free rein on the embedding? (3) How do we compare to a state-of-the-art one-shot imitation learning method? (4) How is our performance affected as we go from one-shot to many-shot? (5) Does this method apply to sim-to-real?

We begin by presenting results from two simulated experimental domains that were put forward for MIL [Finn et al., 2017b]. We then continue to present results for our own experiment where we perform a placing task using a real-world robot arm, similar to that of MIL's third experimental domain. All 3 experiments are shown in Figure 5.4. For all experiments, we ensure that our control network follows a similar architecture to MIL [Finn et al., 2017b] in order to allow fair comparison. In all cases the task-embedding network and control network use a convolutional neural network (CNN), where each layer is followed by layer normalisation [Ba et al., 2016] and an elu activation function [Clevert et al., 2016], except for the final layer, where the output is linear for both the task-embedding and control network. Optimisation was performed with Adam [Kingma and Ba, 2015] with a learning rate of 5×10^{-4}, a batch size of 64, and lambdas were set as follows: $\lambda_{emb} = 1.0$, $\lambda_{ctr}^{U} = 0.1$, $\lambda_{ctr}^{Q} = 0.1$.

Our approach only uses visual information for the demonstrations whilst MIL reports results where the input demonstrations are given with and without actions and robot arm state. For completeness, we have reported all of MIL's results, but our aim is to compare against the results where only visual information is used for input demonstrations. Qualitative results for our approach can be seen in the video[1].

5.4.1 Simulated Reaching

The aim of this first experimental domain is to reach a target of a particular colour in the presence of two distractors with different colours. Input to the control network consist of the (current) arm joint angles, end-effector position, and the 80×64 RGB image, whilst the task-embedding network receives only images (first and last). For details regarding data collection, we point the reader to the Appendix of [Finn et al., 2017b]. Our results (presented in Table 5.1) show that we outperform MIL by a large margin, as well as other variations of our approach. The results show that the embedding loss is vital for the high success rate, with the exclusion leading to a drop in success of over 70%. In addition to the embedding loss, the inclusion of the support loss heavily assists the network in learning the task. Note that it is possible to achieve 33% on this task by randomly choosing one target to reach for. We believe this is an important note, as it appears that MIL is not capable of learning from one visual

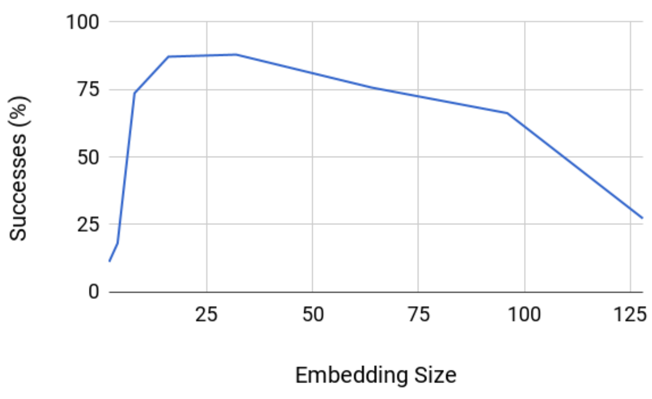


Figure 5.5: How the percentage of success changes as the size of the embedding varies for the simulated reaching domain.

[1] https://sites.google.com/view/task-embedded-control

demonstration alone on this task, resulting in a policy that randomly selects a colour to reach for. As with MIL, only 1-shot capabilities were explored for this domain.

We also use this experimental domain to see how the embedding size effects the performance, and we show the results in Figure 5.5. By increasing the embedding size, we are increasing the dimensionality of our vector space, allowing a greater number of tasks to be learned. But as Figure 5.5 shows, increasing the dimensionality can lead to poor performance. We hypothesise that increasing the embedding size too much can lead to a trivial embedding that does not look for similarities and will thus generalise poorly when encountering new tasks. A balance must be struck between the capacity of the embedding and the risk of overfitting. Although this is an extra hyperparameter to optimise for, Figure 5.5 encouragingly suggest that this can take on a wide range of values.

For these experiments, the CNN consisted of 3 strided convolution layers, each with 40 (3×3) filters, followed by 4 fully-connected layers consisting of 200 neurons. Input consists of a 80×64 RGB image and the robot proprioceptive data, including the arm joint angles and the end-effector position. The proprioceptive is concatenated to the features extracted from the CNN layers of the control network, before being sent through the fully-connected layers. The output of the embedding network is a vector of length 20. The output corresponds to torques applied to the two joints of the arm. The task is considered a success if the end-effector comes within 0.05 meters of the goal within the last 10 timesteps. Further information regarding this task can be accessed from [Finn et al., 2017b].

5.4.2 Simulated Pushing

The second experimental domain from [Finn et al., 2017b] involves a simulated robot in a 3D environment, where the action space is 7-DoF torque control. The goal is to push a randomly positioned object to the red target in the presence of another randomly positioned distractor, where the objects have a range of shapes, sizes, textures, frictions, and masses. The control network input consists of a 125×125 RGB image and the robot joint angles, joint velocities, and end-effector pose, whilst the task-embedding network again receives images only. For details regarding data collection, we point the reader to the Appendix of [Finn et al., 2017b]. In both the 1-shot and 5-shot case, our method surpasses MIL when using its few-shot ability on visual data alone. Unlike the previous experiment, excluding the support loss is less detrimental and leads to better results than MIL in both the 1-shot and 5-shot case.

For these experiments, the CNN consisted of 4 strided convolution layers, each with 16 (5×5) filters, followed by 3 fully-connected layers consisting of 200 neurons. Input consists of a 125×125 RGB image and the robot proprioceptive data, including the joint angles, joint velocities, and end-effector pose. The proprioceptive is concatenated to the features extracted from the CNN layers of the control

	Method	Success (%)
1-Shot	MIL (vision+state+action)	93.00
	MIL (vision)	**29.36***
	Ours (vision)	**86.31**
	Ours ($\lambda_{ctr}^{U} = 0$)	25.68
	Ours ($\lambda_{emb} = 0$)	10.48
	Ours ($\boldsymbol{s} = \vec{0}$)	20.30
	Ours (contextual)	19.17

(a) Simulated Reaching Results

	Method	Success (%)
1-Shot	MIL (vision+state+action)	85.81
	MIL (vision+state)	72.52
	MIL (vision)	**66.44**
	Ours (vision)	**77.25**
	Ours ($\lambda_{ctr}^{U} = 0$)	70.72
	Ours ($\lambda_{emb} = 0$)	58.56
	Ours ($\boldsymbol{s} = \vec{0}$)	02.49
	Ours (contextual)	37.61
5-Shot	MIL (vision+state+action)	88.75
	MIL (vision+state)	78.15
	MIL (vision)	**70.50**
	Ours (vision)	**80.86**
	Ours ($\lambda_{ctr}^{U} = 0$)	72.07
	Ours($\lambda_{emb} = 0$)	67.12
	Ours ($\boldsymbol{s} = \vec{0}$)	-
	Ours (contextual)	-

(b) Simulated Pushing Results

Table 5.1: The result for both simulated reaching (a) and simulated pushing (b), for both our full solution, and a series of ablations. In the tables, $\lambda_{ctr}^{U} = 0$ refers to excluding the support loss, $\lambda_{emb} = 0$ refers to excluding the embedding loss, $\boldsymbol{s} = \vec{0}$ refers to ignoring the output of the embedding, and instead passing in a zero sentence, and '*contextual*' refers to ignoring the output of the embedding, and passing in one of the support example's images directly to the control network. There is no entry for the final 2 rows of *Table (b)* as these are equivalent to their 1-shot counterpart. *Note that in *Table (a)*, the results reported here for MIL were not reported in the paper, and so the results here are reported from running their publicly available code.

network, before being sent through the fully-connected layers. The output of the embedding network is a vector of length 20. The output of the control network corresponds to torques applied to the 7 joints of the arm. The task is considered a success if the robot pushes the centre of the target object into the red target circle for at least 10 timesteps within 100-timestep episode. Further information regarding this task can be accessed from [Finn et al., 2017b].

5.4.3 Real-world Placing via Sim-to-Real

The final experiment tests how well our method works when applied to a real robot. Not only that, but we also look at the potential of our method to be used in a sim-to-real context; where the goal is to learn policies within simulation and then transfer these to the real world with little or no additional training (we focus on the latter). This is an attractive idea, as data collection in the real world is often cumbersome and time consuming.

For these experiments, the CNN consisted of 4 strided convolution layers, each with 16 (5×5) filters, followed by 4 fully-connected layers consisting of 100 neurons. Input consists of a 125×125 RGB image and the robot proprioceptive data, including just the joint angles. The proprioceptive is concatenated to the features extracted from the CNN layers of the control network, before being sent through the fully-connected layers. The output of the embedding network is a vector of length 20. The output of the control network corresponds to torques applied to the 7 joints of a Kinova Mico 7-DoF arm. There is also an additional auxiliary end-effector position output that is learned via an $L2$ distance between the prediction and the ground truth during simulation training. The task is considered a success if the robot drops the held object into the correct target container.

As a note, we also experimented with using a U-Net architecture [Ronneberger et al., 2015] for the control network, where the sentence is concatenated to the image features at the bottleneck, but our experiments showed that channel-wise concatenation at the input layer of the control network worked just as well.

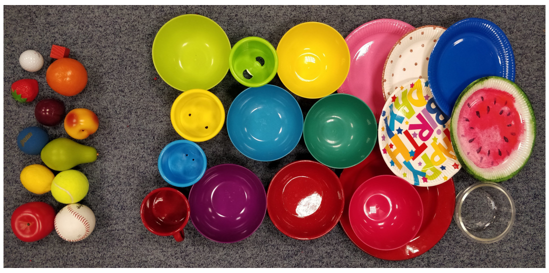


Figure 5.6: The real world test set for the placing domain. Holding objects on the left and placing objects (consisting of bowls, plates, cups, and pots) on the right.

We run a robotic placing experiment much like the one proposed in MIL, where a robot arm is given the task of placing a held object into a specified container whilst avoiding 2 distractors. The key difference is that our data is collected in simulation rather than real world. As summarised in Figure 5.1, our TecNet is trained in simulation with a dataset of 1000 tasks with 12 examples per task. Our containers consist of a selection of 178 bowls from the ShapeNet database [Chang et al., 2015]. To enable transfer, we use domain randomisation; a method that is increasingly being used transfer learned polices from simulation to the real world [James et al., 2017a, Tobin et al., 2017a, Matas et al., 2018]. We record RGB images of size 160×140 from an external camera positioned above the robot arm, joint angles, and velocities along a planned linear path for each example. During domain randomisation, we vary the lighting location, camera position, table texture, target object textures and target object sizes, and create procedurally generated holding objects. An example of the randomisation can be seen in Figure 5.1.

Once the network has been trained, we randomly select one holding object and 3 placing targets from our test set of real-world objects (shown in Figure 5.6); these objects have not been seen before in either simulation or real world. The robot is shown a single demonstration via human teleoperation using the HTC Vive controller. During demo collection, only RGB images and joint angles are collected. A trial is successful if the held object lands in or on the target container after the gripper ihas opened.

One-shot success rates in the real-world is **72.97%**, and is based on 18 tasks with 4 examples each (72 trials total), showing that we are able to successfully cross the reality-gap and perform one-shot imitation learning. The majority of our failure cases appeared when the target objects were cups or plates, rather than bowls. We imagine this is due to the fact that our training set only consisted of bowls. Results for the real world evaluation can be seen in the video[2].

5.4.4 Sim-to-Real Embedding Visualisation

In this section we show t-SNE [Maaten and Hinton, 2008] visualisation of the learnt embedding of the real-world placing task of Section 5.4.3. Note that the TecNet was trained entirely in simulation without having seen any real-world data. In order to visualise how the embedding looks on real-world data, we collect a dataset of 164 tasks, each consisting of 5 demonstrations. Each demonstration consists of a series of RGB images that were collected via human teleoperation using the HTC Vive controller. Each demonstration in these visualisations are represented via the final frame of that demonstration.

[2] `https://sites.google.com/view/task-embedded-control`

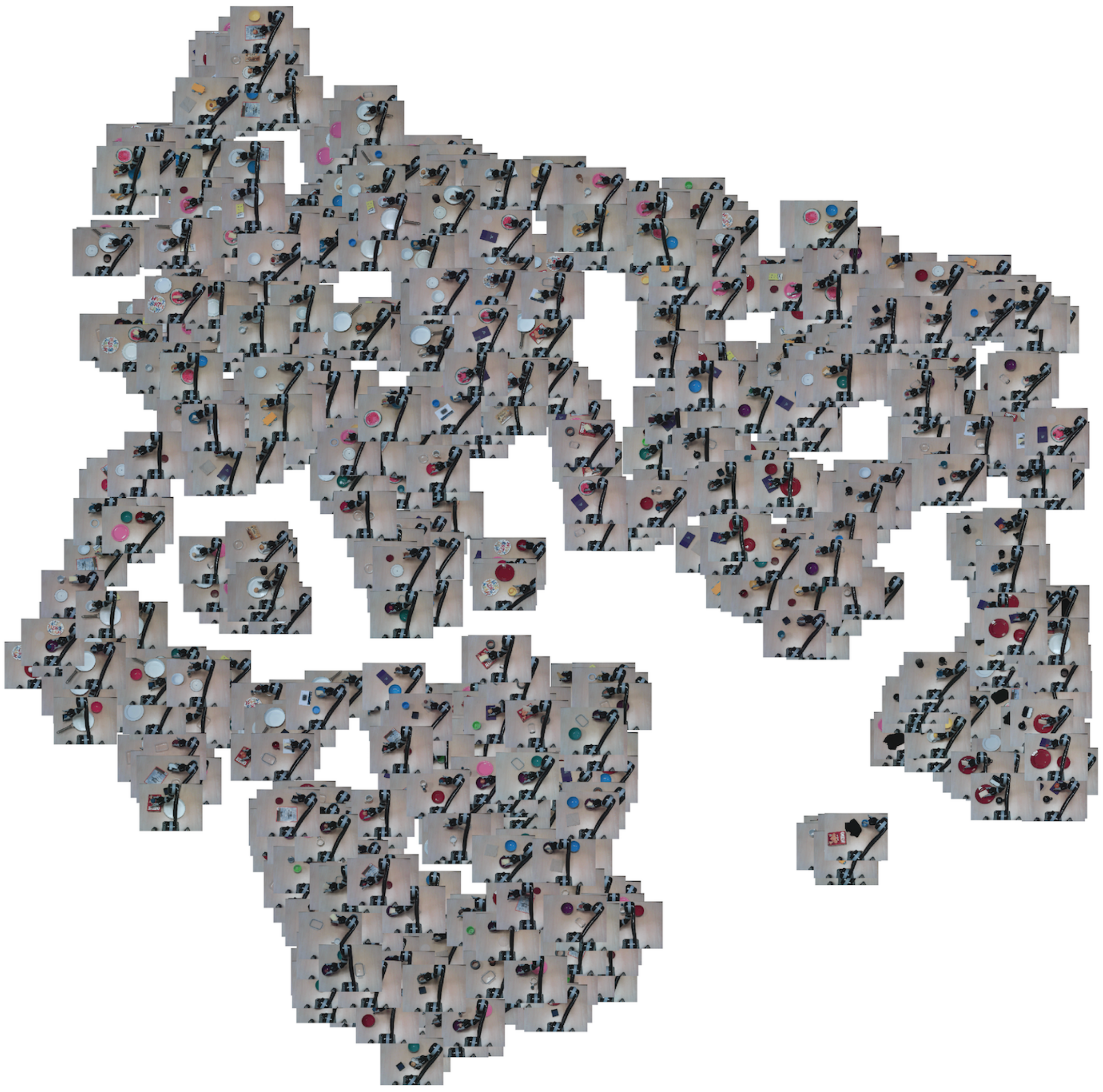


Figure 5.7: A t-SNE visualisation of the individual sentences of each of the demonstrations learnt by the task-embedding network. We embed 5 demonstrations (without averaging) across each of the 164 tasks. The aim of the visualisation is to illustrate how examples of the same task relate with each other. The result shows that the task-embedding network does indeed learn to place examples of the same tasks next to each other, whilst also placing other, visually similar, tasks nearby.

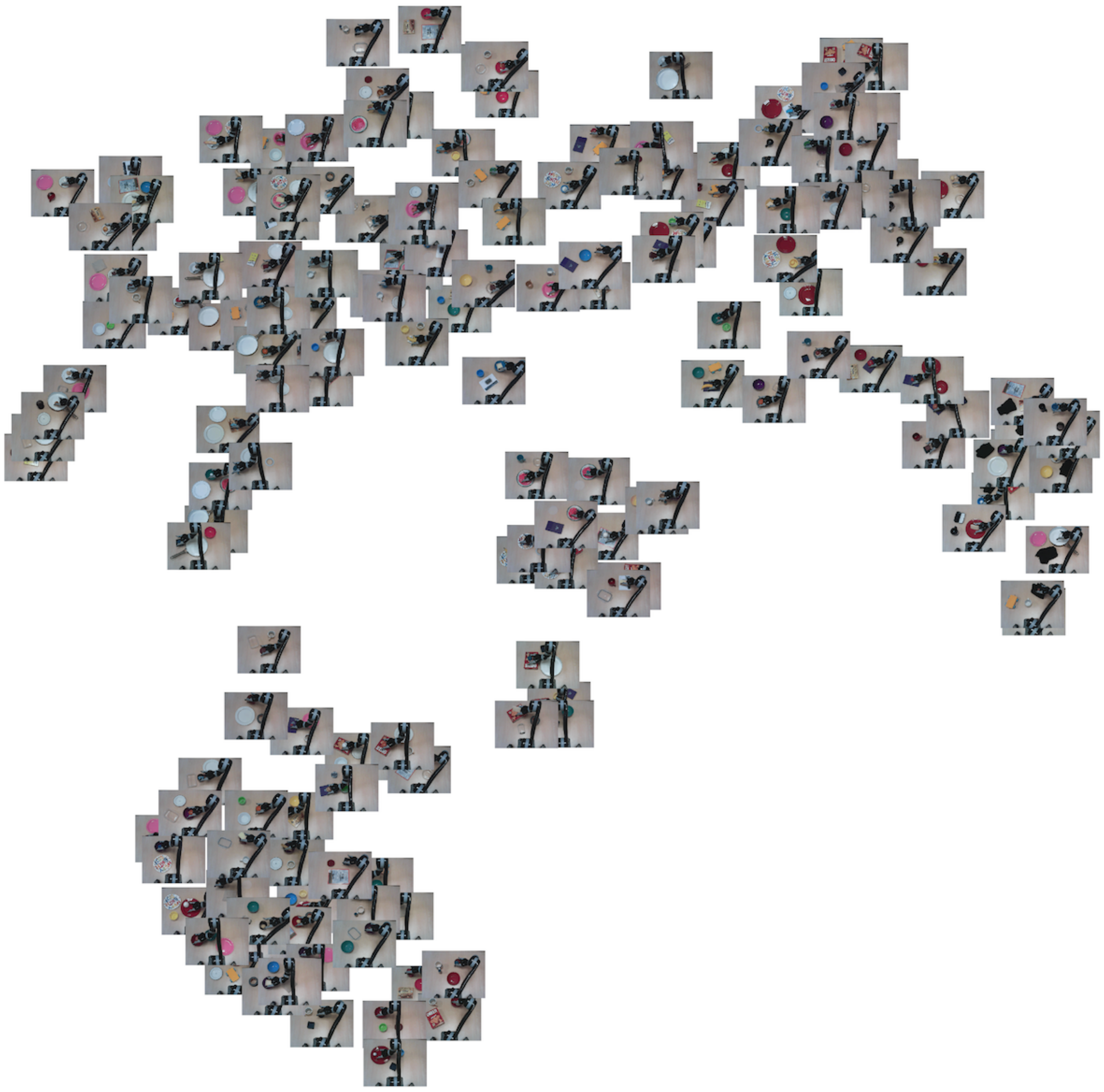


Figure 5.8: A t-SNE visualisation of the combined sentences learnt by the task-embedding network. We embed 5 demonstrations and average to get the task sentence for each of the 164 tasks. Given that we are only plotting the combined sentences, this can be seen as a more legible version of Figure 5.7, focusing on how tasks relate to other tasks, rather than how examples of the same task relate with each other.

5.5 Learning from Human Videos

Humans are able to learn how to perform a task by simply observing their peers performing it once; this is a highly desirable behaviour for robots, as it would allow the next generation of robotic systems, even in households, to be easily taught tasks, without additional technology or long interaction times. Endowing a robot with the ability to learn from a single human demonstration rather than through teleoperation, would allow for a more seamless human-robot interaction.

Previous work has investigated hand-engineered systems which track movements and specify a mapping between the human and robot domains [Lee et al., 2013, Yang et al., 2015]. Rather than explicitly hand-engineered systems, an emerging trend in robotics is to instead learn control directly from raw sensor data in an end-to-end manner. These systems operate well when close and complicated coordination is required between vision and control [Levine et al., 2016a].

Visuospatial skill learning (VSL) has looked at achieving desired goal configuration of objects relative to one another [Ahmadzadeh et al., 2013], and has been demonstrated on an alphabet ordering task, an animal puzzle task, and a Tower of Hanoi task. However, this approach has only been shown with a singe demonstration, which opens up the possibility for ambiguity when giving a demonstration for a task. For example, imagine a blue bowl adjacent to a plate, and a demonstration is given of a person placing an apple into the bowl. There is inherent ambiguity in this single demonstration, as the demonstrator may not want the robot to put the apple into this specific blue bowl, but instead by trying to demonstrate that the apple should be put into any blue container, or that the apple should be put into a container that is adjacent to the plate. It is therefore vital that a method has the ability to take in additional human demonstrations in situations like this to resolve any ambiguity.

Domain-Adaptive Meta-Learning (DAML) is a recent approach that uses an end-to-end method for one-shot imitation of humans [Yu et al., 2018] which leveraged a large amount of prior meta-training data collected for many different tasks. This approach required thousands of examples across many tasks during meta-training: these examples are videos of a person physically performing the tasks and teleoperated robot demonstrations, meaning that there has to be an active and long human presence when collecting the dataset. Is there a way to reduce, or even eliminate, the amount of human presence that is needed when collecting datasets that require footage of humans? We propose that the recent successes in visual simulation-to-reality transfer [James et al., 2017a, Matas et al., 2018, Bousmalis et al., 2018, James et al., 2019b] suggest there is a way.

To that end, we present an approach to the one-shot human imitation learning problem which does not require an active manual intervention during training, thus saving tens or hundreds of researchers

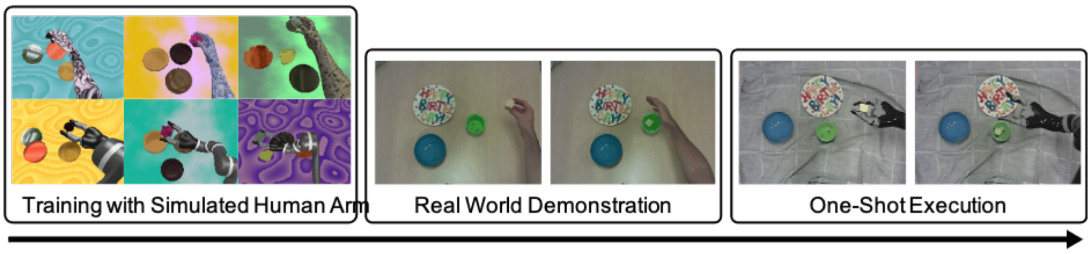


Figure 5.9: The robot gains its ability to infer actions from humans in simulation, and can then learn a new task from a single human demonstration in the real-world.

hours. We show that Task-Embedded Control Networks (TECNets) can be used to infer control policies by embedding human demonstrations that can condition a control policy and achieve one-shot imitation learning. Rather than using real humans to supply demonstrations during training, we instead leverage domain randomisation in an application that has not been seen before: sim-to-real transfer on humans.

Similarly to the work in Chapter 3 on sim-to-real for behaviour cloning, we can learn the control policy by constructing a dataset of both the simulated human and robot trajectories that are generated via inverse kinematics (IK) in the Cartesian space. These can then be used to train a reactive neural network controller which continuously accepts images along with joint angles, and outputs motor velocities.

After training, we are able to deploy a system in the real world which can perform a previously unseen task in a new configuration after a single real-world human demonstration. Our approach, which is summarised in Figure 5.9, is evaluated on pushing and placing tasks in both simulation and in the real world. We show we are able to achieve similar results to a system trained on real-world data. Moreover, we show that our approach remains robust to visual domain-shifts, such as a substantially different background, between the human demonstrator and the robot agent performing the task.

5.5.1 Method

We expand on the TECNets method introduced in Section 5.3 section by incorporating the notion of a human demonstrator which can be summarised in Figure 5.10. We slightly modify the definition of a task $\mathfrak{T}$ to instead include two collections of examples: a human demonstrator collection $\mathfrak{H} = \{\tau_1^{\mathfrak{H}}, \cdots, \tau_K^{\mathfrak{H}}\}$ and a robot agent collection $\mathfrak{R} = \{\tau_1^{\mathfrak{R}}, \cdots, \tau_K^{\mathfrak{R}}\}$, such that $\mathfrak{T} = (\mathfrak{H}, \mathfrak{R})$, where $\tau^{\mathfrak{R}} = [(\mathbf{o}_1, \boldsymbol{a}_1), \ldots, (\mathbf{o}_T, \boldsymbol{a}_T)]$, similarly to Section 5.3, and $\tau^{\mathfrak{H}} = [\mathbf{o}_1, \ldots, \mathbf{o}_T]$. Note that the length of the human trajectory does not have to equal the length of the corresponding robot trajectory.

From this, we now pick three disjoint sets of examples (rather than the original two) for every task $\mathfrak{T}^j$: a support set of human examples $\mathfrak{H}_U^j$, a query set of human examples $\mathfrak{H}_Q^j$, and a set of robot examples $\mathfrak{R}^j$.

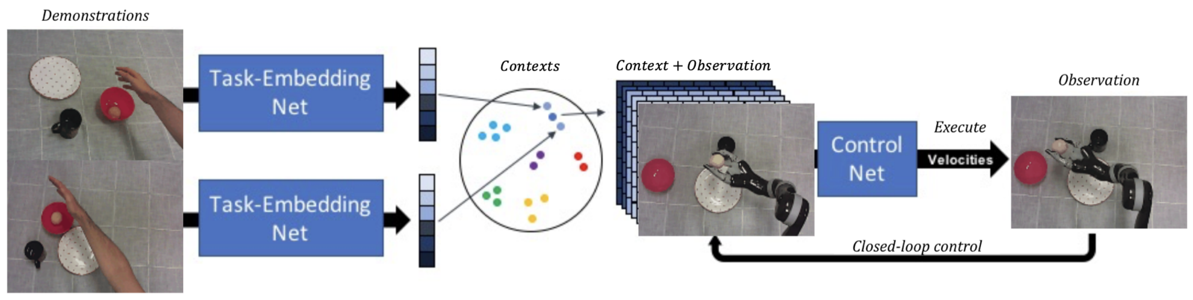


Figure 5.10: We use Task-Embedded Control Networks (TECNets) to allow tasks to be learned from a single human demonstration. Images of human demonstrations are embedded into a compact representation of a task, which is then expanded and concatenated (channel-wise) to the most recent observation from a new configuration of that task before being sent through the control network in a closed-loop manner. Both the task-embedding net and control net are jointly optimised to produce a rich embedding.

In analogy to Equation (5.1) a combined context $\boldsymbol{s}^j \in \mathbb{R}^N$ for a task is computed by taking the normalised mean of the context for each example in the support set of human examples $\mathfrak{H}_U^j$:

$$\boldsymbol{s}^j = \left[\frac{1}{|\mathfrak{H}_U^j|} \sum_{\tau \in \mathfrak{H}_U^j} f_\theta(\tau) \right]^\wedge . \tag{5.5}$$

We then train the embedding model to produce a higher dot-product similarity between human demonstrations of a task's embedded example $f_\theta(\tau)$ and its context $\boldsymbol{s}^j$ than to contexts of human demonstrations from other tasks $\boldsymbol{s}^i$:

$$\mathcal{L}_{emb} = \sum_{\tau \in \mathfrak{H}^j} \sum_{i \neq j} max[0, margin - f_\theta(\tau) \cdot \boldsymbol{s}^j + f_\theta(\tau) \cdot \boldsymbol{s}^i] . \tag{5.6}$$

Additionally, given a context $\boldsymbol{s}^j$, computed from the support set $\mathfrak{H}_U^j$, as well as examples from the robot set $\mathfrak{R}^j$ for the same task we can compute the following behaviour-cloning loss for the policy π:

$$\mathcal{L}_{ctr} = \sum_{\tau \in \mathfrak{R}_Q^j} \sum_{(\mathbf{o},\boldsymbol{a}) \in \tau} \|\pi(\mathbf{o}, \boldsymbol{s}^j) - \boldsymbol{a}\|_2^2 . \tag{5.7}$$

The final loss is the combination of the embedding loss $\mathcal{L}_{emb}$, the control loss on the support set for the human examples, and the control loss on the robot examples:

$$\mathcal{L}_{tec} = \lambda_{emb}\mathcal{L}_{emb} + \lambda_{\mathfrak{H}ctr}^U \mathcal{L}_{\mathfrak{H}ctr}^U + \lambda_{\mathfrak{R}ctr}\mathcal{L}_{\mathfrak{R}ctr} . \tag{5.8}$$

Note that only the human examples of the same task are explicitly enforced to be close together in the embedding space, rather than human and robot examples. Although we could have also enforced an additional embedding loss on human and robot examples being close together, in practice we found that this was not necessary. This is due to the joint training of both task-embedding and control networks which enforces the network to implicitly learn to map the embedded human examples to a set of corresponding robot actions. Pseudocode for both the training is provided in Algorithm 3.

Algorithm 3 Training loss computation for one batch. B is the batch size, K_U and K_Q are the number of examples from the support and query set respectively, $K_{\mathfrak{R}}$ is the number of robot examples to sample, and $RandomSample(S, N)$ selects N elements uniformly at random from the set S.

```
1:  procedure TRAINING ITERATION
2:      $\mathcal{B} = RandomSample(\{\mathfrak{T}_1, \cdots, \mathfrak{T}_N\}, B)$
3:      for $\mathfrak{T}^j \in \mathcal{B}$ do
4:          $(\mathfrak{R}^j, \mathfrak{H}^j) \leftarrow \mathfrak{T}^j$
5:          $\mathfrak{H}^j_U = RandomSample(\mathfrak{H}^j, K_U)$
6:          $\mathfrak{H}^j_Q = RandomSample(\mathfrak{H}^j \backslash \mathfrak{H}^j_U, K_Q)$
7:          $\mathfrak{R}^j_{\prime} = RandomSample(\mathfrak{R}^j, K_{\mathfrak{R}})$
8:          $\boldsymbol{s}^j_U = \left[\frac{1}{K_U} \sum_{\tau \in \mathfrak{H}^j_U} f_\theta(\tau)\right]^{\wedge}$
9:          $\boldsymbol{s}^j_q = f_\theta(\tau_q) \quad \forall \tau_q \in \mathfrak{H}^j_Q$
        $\mathcal{L}_{emb} = \mathcal{L}^U_{\mathfrak{H}_{ctr}} = \mathcal{L}_{\mathfrak{R}_{ctr}} = 0$
10:     for $\mathfrak{T}^j \in \mathcal{B}$ do
11:         $\mathcal{L}_{emb} \mathrel{+}= \sum_q \sum_{i \neq j} max[\,0, margin - \boldsymbol{s}^j_q \cdot \boldsymbol{s}^j_U + \boldsymbol{s}^j_q \cdot \boldsymbol{s}^i_U]$
12:         $\mathcal{L}^U_{\mathfrak{H}_{ctr}} \mathrel{+}= \sum_{\tau \in \mathfrak{H}^j_U} \sum_{(\mathbf{o}, \boldsymbol{a}) \in \tau} \|\pi(\mathbf{o}, \boldsymbol{s}^j_U) - \boldsymbol{a}\|^2_2$
13:         $\mathcal{L}_{\mathfrak{R}_{ctr}} \mathrel{+}= \sum_{\tau \in \mathfrak{R}^j_{\prime}} \sum_{(\mathbf{o}, \boldsymbol{a}) \in \tau} \|\pi(\mathbf{o}, \boldsymbol{s}^j_U) - \boldsymbol{a}\|^2_2$
14:     $\mathcal{L}_{tec} = \lambda_{emb}\mathcal{L}_{emb} + \lambda^U_{\mathfrak{H}_{ctr}}\mathcal{L}^U_{\mathfrak{H}_{ctr}} + \lambda^Q_{\mathfrak{R}_{ctr}}\mathcal{L}^Q_{\mathfrak{R}_{ctr}}$
15:     return $\mathcal{L}_{tec}$
```

Input to the task-embedding network consists of $(width, height, 3 \times |\tau|)$, where 3 represents the RGB channels. As in the TECNets work, we found that we only need to take the first and last frame of an example trajectory τ for computing the task embedding and so we discard intermediate frames, resulting in an input of $(width, height, 6)$. The context from the task-embedding network is then tiled and concatenated channel-wise to the input of the control network (as shown in Figure 5.10), resulting in an input image of $(width \times height \times 3 + N)$, where N represents the length of the embedding.

Data Collection in Simulation

Many approaches to human imitation rely on training in the real world. This has many disadvantages, but most evident is the amount of time and effort needed to collect data for the training dataset. In the case of DAML, thousands of demonstrations had to be recorded, which rely on an active human presence to obtain both human and robot demonstrations, as the robot still has to be controlled in some way. For instance, in the DAML placing experiment a total of 2586 demonstrations were collected to form the training dataset, meaning tens of research hours dedicated to collecting data, with no guarantees that the dataset allows the network to generalise well enough. Training in simulation provides much more flexibility and availability of data: data generation can be easily parallelised and does not require constant human intervention. Additionally, there have been many successful examples of systems trained in simulation and then run in the real-word; one common approach to do this is domain randomisation [Sadeghi and Levine, 2017, Tobin et al., 2017a, James et al., 2017a, Matas et al., 2018, Bousmalis et al.,

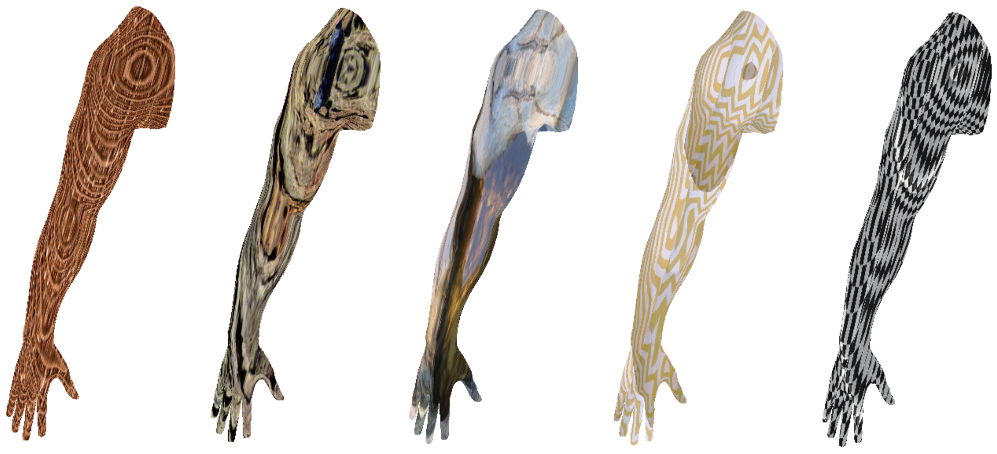


Figure 5.11: An example of the variations we get when we apply domain randomisation to the simulated human arm.

2018, James et al., 2019b].

Our approach generates the training dataset using PyRep [James et al., 2019a], a recently released robot learning research toolkit, built on top of CoppeliaSim/V-REP [E. Rohmer, 2013]. We modelled a 3D mesh of a human arm from nonecg.com, which we then broke down into rigid shapes. Our simulated arm has 26 degrees of freedom: 3 in the shoulder, 2 in the elbow, 2 for the wrist and the remaining 19 in the hand. 26 revolute joints link together the different rigid shapes: to emulate the soft-body behaviour of a real arm during motion, adjacent shapes slide over each other, making previously hidden parts of each shape visible. The resulting effect is very similar to real human arm motion.

Following previous work in behaviour cloning [James et al., 2017a], during dataset generation we collect a series of linear paths generated in the Cartesian space via inverse kinematics (IK) in order to construct the task sequence. At each timestep, we collect the image, the joint angles and the joint velocities at each timestep for both human arm and robot. To achieve sim-to-real transfer we perform domain randomisation. Specifically, we sample from a set of 5000 textures and procedurally generated images (via Perlin noise), and apply them to all objects in the scene and to the human arm (an example can be seen in Figure 5.11). Additionally, we sample the position, the orientation and the size of the objects from a uniform distribution. The starting configuration of both the demonstrator and the agent, camera pose, light directions and lighting parameters are sampled from a normal distribution. A snapshot of the simulation and real-world setup can be seen in Figure 5.12.

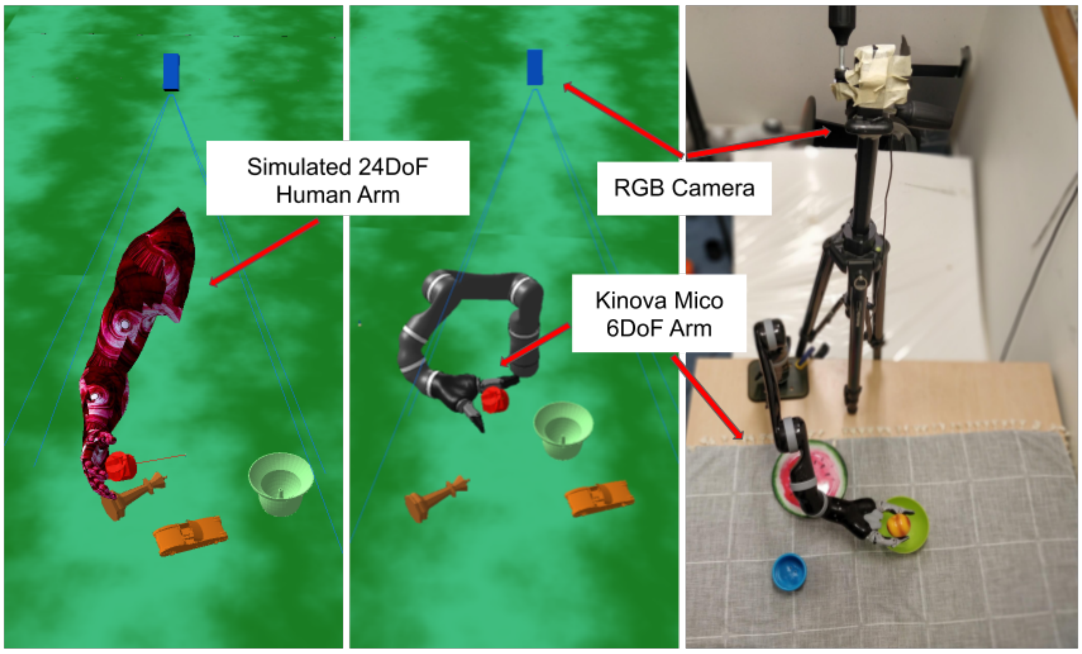


Figure 5.12: The simulation and real world setup. On the left, we see the 24 DoF arm, and in the centre we see the 6 DoF Kinova Mico arm; both have domain randomisation applied. On the right, we see real-world setup with the Kinova Mico. Observations come from an over-the-shoulder RGB camera in both simulation and the real world.

Training

Our task-embedding network and control network use a convolutional neural network (CNN), which consists of 4 convolution layers, each with 16 filters of size 5×5, followed by 3 fully-connected layers consisting of 200 neurons. Each layer is followed by layer normalisation [Ba et al., 2016] and an elu activation function [Clevert et al., 2016], except for the final layer, where the output is linear for both the task-embedding and control network.

Input consists of a 125×125 RGB images and the robot proprioceptive data, including the joint angles. The proprioceptive data is concatenated to the features extracted from the CNN layers of the control network, before being sent through the fully-connected layers. The output of the embedding network (embedding size) is a vector of length 20. The output of the control network corresponds to velocities applied to the 6 joints of a Kinova Mico 6-DoF arm. During training, we set the margin to be 0.1 for the embedding loss $\mathcal{L}_{emb}$, and set both the support and query size to be 5.

Optimisation was performed with Adam [Kingma and Ba, 2015] with a learning rate of 5×10^{-4} and a batch size of 100. Lambdas were set as follows: $\lambda_{emb} = 0.1$, $\lambda^{U}_{\mathfrak{H}_{ctr}} = 1.0$, and $\lambda_{\mathfrak{R}_{ctr}} = 1.0$.

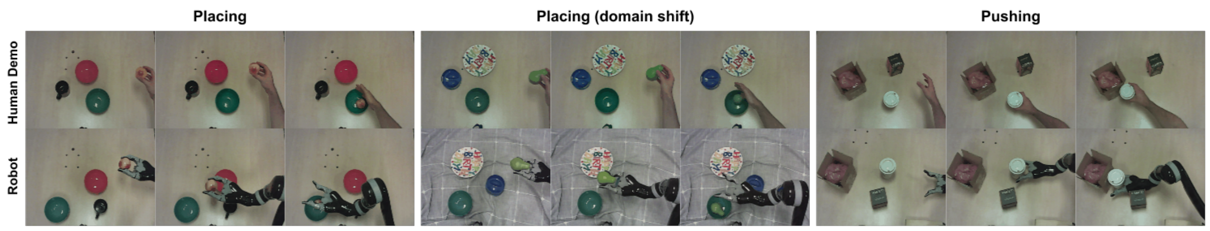


Figure 5.13: The three tasks that we evaluate our model on in the real world. Left: placing an object in one container with two distractor, same camera pose for human and robot and same background. Centre: the same placing task as on the left, but with a domain shift (cloth on the table) between the human demonstration and the robot execution. Right: pushing one object to a target with one distractor, same camera pose for human and robot and same background.

5.5.2 Results

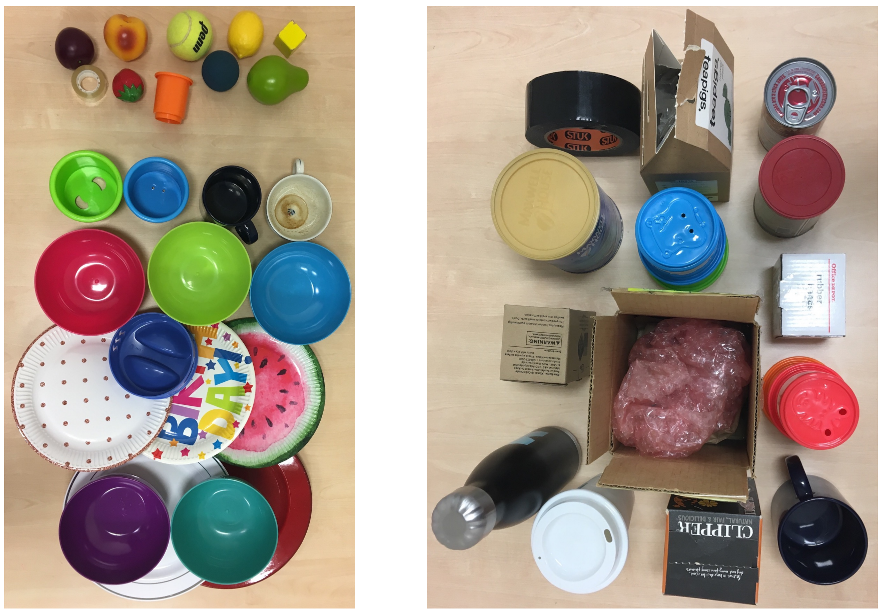
In our experiments we try to answer the following questions: (1) Can TECNets learn the domain shift between a demonstrator and an agent? In other words, can our approach learn an embedding of a task given demonstrator examples, and also a mapping from the demonstrator domain to the agent domain for control? (2) Is it possible to learn a task from a real-world human demonstration when all the training is done is simulation? (3) How does our approach compare to another state-of-the-art one-shot human imitation learning method? We consider two experiments, placing and pushing, which were undertaken for DAML [Yu et al., 2018] in order to compare our approach with their results. We run a set of experiments in both simulation and in the real world.

Placing

We begin by presenting our results for the placing experiment, both in simulation and in the real world: the goal is to place a hand-held object in a container, with other two containers in the scene acting as distractors. A trial is successful if the object lands inside the container. Our dataset features a total of 2280 tasks, where each contains 15 simulated human demonstrations, and 15 simulated robot demonstrations. For each task we sampled three objects from the MIL dataset [Finn et al., 2017b] of 105 training meshes, and used them as target containers and distractors; we randomised the scene as described in 5.5.1 and we trained the network in simulation with the parameters in 5.5.1.

We evaluated one-shot placing in simulation on 74 tasks, with 6 trials each, using the MIL test meshes: in every trial we randomise the position of the objects and of the camera, and we procedurally generate the hand-held object. We also performed evaluation for the same system in the real world (Figure 5.13 Left) on 18 tasks and 4 trials, using the containers and the held objects shown in Figure 5.14a, maintaining the same camera pose and background between demonstration and trial.

The results for the placing experiment are shown in Table 5.2. We find that the robot is able to learn



(a) Placing (b) Pushing

Figure 5.14: The real world test set for both the placing (a) and pushing (b) domain. For the placing domain (a), holding objects are on the top and placing objects (consisting of bowls, plates, cups, and pots) are on the bottom.

from just one human demonstration of a previously unseen task, and can leverage the training with domain randomisation to bridge the reality gap, with comparable success rates to DAML. Additionally, we report the results of simulated placing evaluation for a network trained without the control loss on the human examples support set $\mathcal{L}^{U}_{\mathfrak{H}ctr}$. As it was previously outlined in *James at al.* [James et al., 2018], the inclusion of the support loss assists the network in learning the task and the mapping between domains. In addition, *James at al.* [James et al., 2018] also showed that when setting $\lambda_{emb} = 0$, performance degrades catastrophically, emphasising that the embedding network is learning a meaningful representation for the control network. When transferring from simulation to reality we get an expected drop from 94.1% to 88.9% (as is common in sim-to-real related work [James et al., 2017a, Bousmalis et al., 2018, James et al., 2019b]), which falls below DAML's 93.8%. Given the difference in robot, sensors, objects, and training setups, it is difficult to compare approaches evenly, but we believe this shows that our method can get comparable performance despite using no real-world data.

We also report the results of a real world experiment with a dataset where we simply randomised the scene and made the held object float to the target bowl, without using our simulated human arm. The results show that without the simulated arm, the resulting real-world policy chooses a target at random; therefore the arm model is necessary to successfully learn to imitate from a single human demonstration.

	Placing Experiment
DAML: Real World	93.8%
Ours: Real World	**88.9%**
Ours: Sim	94.1%
Ours: Sim $\lambda^U_{ctr} = 0$	78%
Ours: Real World (No Sim Arm)	39%

Table 5.2: One-shot success rate of the placing experiment, using novel objects. In simulation the scene is randomised for every trial. In the real-world the human demonstrations are taken from the perspective of the robot. We achieve comparable performance to DAML despite training on no real-world data.

	Pushing Experiment
DAML: Real World	88.9%
Ours: Real World	**84.7%**
Ours: Sim	87.6%

Table 5.3: One-shot success rate of the pushing experiment. In simulation the scene is randomised for every trial, whereas in the real-world the human demonstrations are taken from the perspective of the robot. As in the placing results, we achieve comparable performance to DAML despite training on no real-world data.

Pushing

In the pushing experiment the goal is to push an object against a target amid one distractor: a trial is successful if the object hits or falls within 5cm of the target. Our dataset features a total of 1620 tasks, with 15 domain randomised demonstrations for both robot and human, using objects from the MIL dataset.

We evaluated one-shot pushing in both simulation and real-world (Figure 5.13 Right), with the same number of trials as for the previous placing experiments. The objects used for the real-world experiment are shown in Figure 5.14b. We report the results in Table 5.3 together with the DAML results to show that our network trained in simulation has once again comparable performance to a model trained with real data.

Large Domain Shift

As a final experiment, we tested how resilient our model is against large domain shifts in the real world, expecting it to leverage the adaptability acquired from domain randomisation. We evaluated placing in the real world taking the human demonstrations with a cloth on the table, and then making the robot perform the same task with the table cloth removed, therefore with a substantial change of background (Figure 5.13 Centre): the model placed the held object correctly **87.5% of the 72 trials**.

We have therefore shown that due to domain randomisation our performance does not degrade on large domain shifts, whereas for example in DAML the success rate under large change of scenes drops by up to 15%. This showcases the benefits of leveraging large scale simulations for robotic learning.

Failure Modes

Across both placing and pushing experiments, we found two common failure modes: (1) **failure from task identification**, where the robot successfully executed a task, but not the one that was specified; for example, in the pushing task, correctly pushing the distractor to the target rather than the desired object. (2) **Failure from control**, where the robot (appeared) to identify the task, but did not succeed; for example, in the placing task the robot would head towards the correct object, but would overshoot or undershoot the target.

5.6 Conclusion

We have presented TECNets, a powerful few-shot learning approach for end-to-end few-shot imitation learning. The method works by learning a compact description of a task via an embedding network, that can be used to condition a control network to predict action for a different example of the same task. Our approach is able to surpass the performance of MIL [Finn et al., 2017b] for few-shot imitation learning in two experimental domains when only visual information is available. Unlike many other meta-learning approaches, our method is capable of continually learning new tasks without forgetting old ones, and without losing its few-shot ability. Moreover, we demonstrate that the few-shot ability can be trained in simulation and then deployed in the real world. Once deployed, the robot can continue to learn new tasks from single or multiple demonstrations.

Similar to other meta-learning approaches, we expect the approach to perform poorly when the new task to learn is drastically different from the training domain; for example, a TECNet trained to place items in containers would not be expected to learn few-shot pushing. Having said that, if the training set were to include a wide range of tasks, generalising to a broad range of tasks may be possible, and so this is something that should be looked at further.

In addition, we have presented an approach to the one-shot human imitation problem that leverages zero human interaction during training time. We achieve this by the combination of 2 methods. Firstly, we extending Task-Embedded Control Networks (TECNets) to infer control polices by embedding human demonstrations that can condition a control policy and achieve one-shot imitation learning. Secondly, and most importantly, we show that we are able to perform sim-to-real on humans which allows us to train our system with no real-world data. With this approach, we are able to achieve similar performance to a state-of-the-art alternative method that relies on thousands of training demonstrations collected in the real-world, whilst also remaining robust to visual domain-shifts, such as a substantially different backgrounds.

Although we achieve this performance without real-world data, there are a few limitations. For one, because of our data generation process, which generates linear IK trajectories for both the human arm and the robot, we do not get a diverse range of trajectories in our dataset. This has the consequence that when giving the few-shot demonstrations in the real world, the human may have to (undesirably) perform the demonstrations in a certain way; nevertheless this has a small impact due to our approach taking into consideration only the first and the last frames of the demonstration. Moreover, there is a fundamental question about the limitations of training in simulation itself: while real-world data collection may be time consuming, it can in principle be collected by anyone, whilst setting up a sim-to-real environment requires greater expertise.

For future work, one way to improve data diversity could be to instead obtain the simulated human arm trajectories from a reinforcement learning algorithm, which would offer a greater diversity of behaviours to train the task-embedding network. Moreover, we hope to further investigate the variety of human actions that can be transferred from simulation to reality. For example, in this work, we have shown that a human arm can be transferred, but would the same method work for demonstrations including the entire torso of a human? We hope this work provides the first step in answering this question.

CHAPTER 6

RLBench

Contents

6.1 Introduction

In the previous chapter, we achieved high success on one-shot imitation learning with TECNets. However, one important weakness of the evaluation was the limited variation across tasks. This was predominantly down to the fact that we were in uncharted territory: there did not exist a suitable benchmark or standard set of tasks for the few-shot manipulation community. In fact, despite the large body of work looking at increasing the capabilities of robotic agents through the use of reinforcement learning [James and Johns, 2016a, Kalashnikov et al., 2018], meta-learning [Finn et al., 2017b, James et al., 2018, Yu et al., 2018], multi-task learning [Devin et al., 2017, Hausman et al., 2018], etc, there is currently no standard in place for comparing manipulation methods in these respective areas. Although there exist benchmarks such as OpenAI Gym [Brockman et al., 2016b] and DeepMind Control Suite [Tassa et al., 2018] for evaluating continuous-control reinforcement learning algorithms, their focus is not on real-world problems, and it is often the case that algorithms in these toy-benchmarks do not scale to more complex, real-world tasks. As we have seen, few-shot learning methods for robotics also suffer from a lack of well defined tasks; for example, in Finn *et al.* [Finn et al., 2017b] and our work in the previous chapter, there is a very narrow distribution of tasks, where the task of "placing a peach into a red bowl" would be considered a different task to "placing an apple in to a green bowl". Despite the increase in these data-driven approaches, it is not clear where the ideal location on this 'learning' spectrum lies for complex robotics tasks that we may one day want robots performing in our homes. Given all of these problems, there seems to be a need for a benchmark that evaluates not only the diverse range of robot learning fields that are now emerging, but also a range of visually-guided manipulation approaches from both sides of the spectrum.

This motivates the need for a one-size-fits-all benchmark that allows the capability to utilise large-scale data, whilst also allowing classical systems to be compared. To that end, we present RLBench, which is an ambitious large-scale benchmark and learning environment designed to facilitate research in a number of both classical and deep-learning based robot manipulation areas. RLBench is deliberately highly challenging and forward looking. The benchmark includes 100 completely unique, hand-designed tasks ranging in difficulty (shown in Figure 6.1), which share a common Franka Emika Panda robot arm, featuring a range of sensor modalities, including joint angles, velocities and forces, an eye-in-hand camera and an over-the-shoulder stereo camera setup. Each of the 100 tasks comes with a number of textual descriptions and an infinite set of demonstrations made possible through our task building tools that use waypoint-based motion planning.

In this chapter, we discuss a host of research areas that would benefit from this benchmark, including, but not restricted to, reinforcement learning, imitation learning, few-shot learning, multi-task learning, and the incorporation of explicit model-based intermediate representations from computer vision and SLAM. In addition to the benchmark, we also contribute an open-source set of tools that will allow rapid development of new tasks (through the use of PyRep [James et al., 2019a]) in order to improve the size and scope of the benchmark over time. To summarise, RLBench has the following 3 key aims:

- Provide a benchmark and learning environment for both 'robot learning' and 'traditional' methods.
- Provide the a large-scale few-shot challenge, where given M training tasks and N unseen tasks, a system must take K different demonstrations of each of the N unseen tasks, and then be able to perform these tasks in new configurations.
- Provide a set of tools to allow easy task creation.

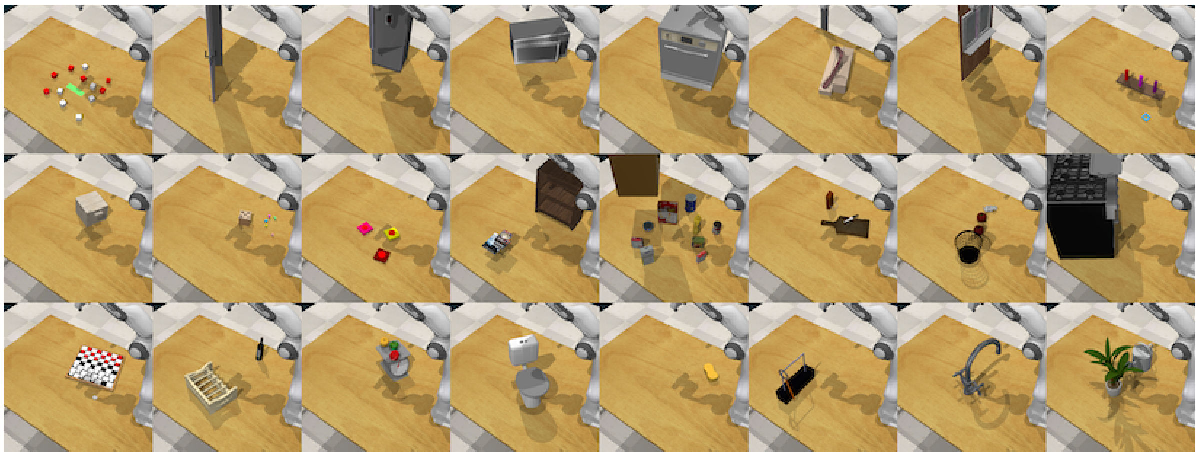


Figure 6.1: RLBench is a large-scale benchmark consisting of 100 completely unique, hand-designed tasks. In this figure we show a sample of 24 tasks that feature in the benchmark. Example tasks include stacking a set of 6 colored blocks in a pyramid (top left), inserting a shape onto a peg (top right), finish setting up a checkers board (bottom left), and watering a plant (bottom right). To get a better understanding of the variety of tasks, please watch the video.

6.2 Related Work

We review existing datasets, benchmarks, and learning environments that could be considered similar to ours in an effort to further motivate RLBench. Firstly we cover reinforcement learning benchmarks, followed by benchmarks designed specifically for manipulation.

Reinforcement Learning Largely as a consequence of the seminal work that saw an algorithm learn to play a range of Atari 2600 video games to superhuman level directly from image pixels [Mnih et al., 2015], deep reinforcement learning (DRL) has increasingly become prevalent in the literature, leading to a number of recent further success in the games of Go [Silver et al., 2016], Chess [Silver et al., 2017], StarCraft [DeepMind, 2019], and Dota [OpenAI, 2018]. With the success of these approaches, there has been a surge in developing DRL algorithms to solve continuous control environments [Lillicrap et al., 2015a, Schulman et al., 2015, Schulman et al., 2017, Haarnoja et al., 2018a, Fujimoto et al., 2018a]. These learned (continuous control) agents are usually tested on benchmarks such as OpenAI Gym [Brockman et al., 2016b] or the DeepMind Control Suite [Tassa et al., 2018]. However, apart from a small number of robotic tasks in OpenAI Gym, these benchmarks feature only toy tasks that often do not resemble real-world problems that robots will need to overcome. To combat this, many projects create their own manipulation tasks to evaluate their approach, making comparisons difficult. As a direct consequence of this, these created tasks can often succumb to unintentionally introducing another hyperparameter into the method in the form of the task design itself. For example, a method could fail on a more challenging task, and so results would only be presented for a simpler set of tasks. This is something a standard benchmark of tasks could alleviate. Moroever, the continuous control tasks in both OpenAI Gym [Brockman et al., 2016b] and the DeepMind Control Suite [Tassa et al., 2018] do not provide an easy-to-use task building tool, and so users must define tasks in XML using the MuJoCo interface (which is cumbersome, error prone, and time consuming). RLBench however, is built around a CoppeliaSim/V-REP [Rohmer et al., 2013] and PyRep [James et al., 2019a] interface, allowing for drag-and-drop style scene building, and a whole host of robotic tools, such as inverse/forward kinematics, motion planning, visualisations, etc. RoboNet [Dasari et al., 2019] is a recent real-world dataset consisting of 15 million video frames, collected via a range of robots interacting with different objects in a table-top setting. The aim is to use this dataset to pretrain reinforcement learning models and then transfer knowledge to a different test environment. The aim of RLBench however, is to offer a benchmark platform for a number of research areas. We should also mention the very recently announced Meta-World project [Yu et al., 2019], a multi-task simulated benchmark for meta-learning research in manipulation, which was released concurrently with this work. Although Meta-World has a similar goal to RLBench, our benchmark has significantly more tasks, uses many more sensor modalities, has a visually higher fidelity, and has the ability to generate demonstrations.

Manipulation Most related work in benchmarking robot manipulation algorithms often concentrates on solving only one of the manipulation sub-problems, focusing on either perception, grasping, or planning. But first, we look at benchmarks that evaluate the system as a whole. The Amazon Robotics Challenge (ARC) [Eppner et al., 2016] was an attempt to create a benchmark for robotic picking and stowing. Although it was a successful challenge that drew many conclusions, such as the usefulness of a dual gripper and suction cup end-effector [Morrison et al., 2018], it was difficult to reproduce in a lab setup. The ACRV Picking Benchmark [Leitner et al., 2017] aimed to solve this by creating a similar, but reproducible setup to the ARC. The issue with picking and stowing is that it is but one of many possible tasks; RLBench on the other hand comes with 100 unique tasks, many of which involve some aspect of picking and placing. Similarly to ARC, the RoboCup@Home competition [Wisspeintner et al., 2009] is run annually, but has a greater range of tasks that must be completed. However, given that no large-scale data is given beforehand, this makes reinforcement learning and other end-to-end approaches difficult to apply in the competition. RLBench is a platform that can unify both old and new methods and compare them on an even playing field.

For evaluating imitation learning systems in particular, RoboTurk [Mandlekar et al., 2018] was a recent attempt to leverage crowd sourcing to obtain data for tasks, but because of this the system has only three tasks. Whilst RoboTurk is entirely in simulation, Simitate [Memmesheimer et al., 2019] on the other hand is a hybrid approach, where real world observations (RGB-D camera calibrated against a motion capturing system) are combined with a simulated environment for benchmarking. In contrast

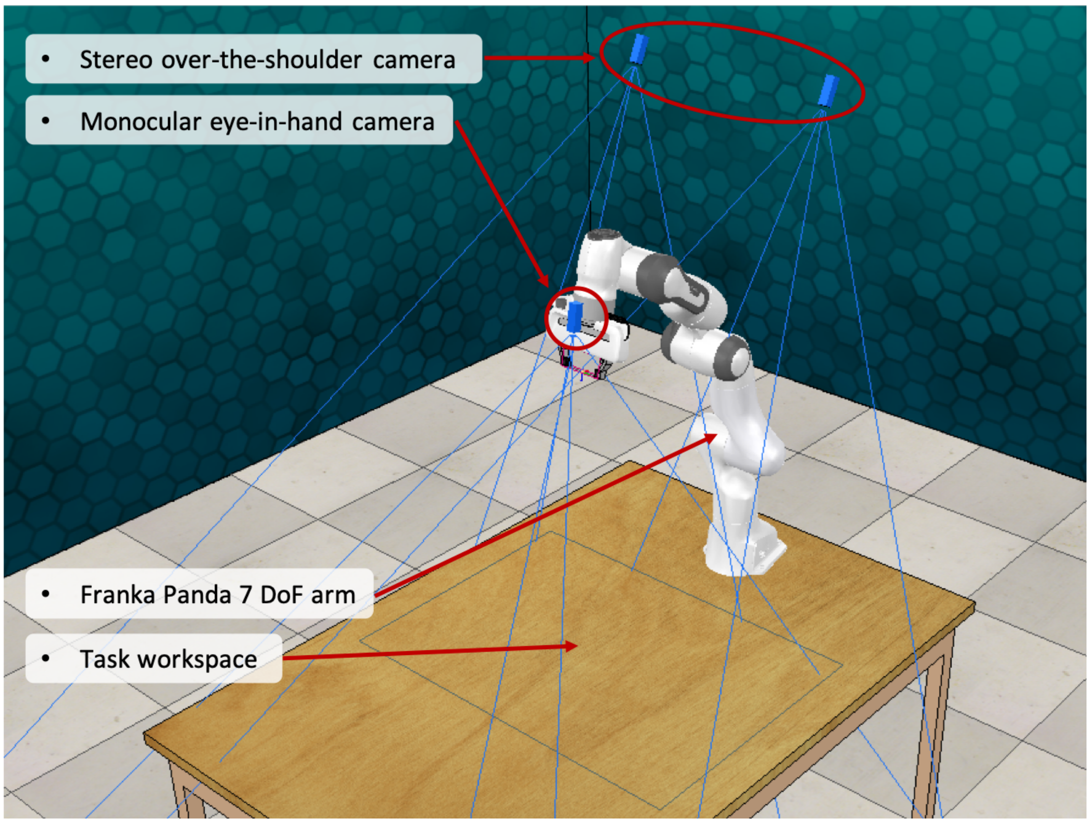


Figure 6.2: The CoppeliaSim/V-REP scene consists of a Franka Panda affixed to a wooden table, surrounded by 3 directional lights. Observations include rgb, depth, and segmentation masks from an over-the-shoulder stereo camera and a eye-in-hand monocular camera, along with robot proprioceptive data, which includes joint angles, velocities, and torques, and the gripper pose. The arm can be easily swapped out for another arm if required.

to RoboTurk, we do not crowd source our demonstrations, but instead rely on an infinite supply of generated demonstrations collected via motion planners. Although Simitate offers the benefit of being partially a real-world dataset, the addition of new tasks requires time-consuming calibration and motion capturing; our system on the other hand sacrifices the real-world aspect, but in exchange we receive the ability generate a diverse range of tasks in a scalable way.

Moving on from whole-system benchmarks, there are a host of benchmarks that focus on sub-problems, for example perception datasets, from both the computer vision community (such as ILSVRC [Russakovsky et al., 2015a], COCO [Lin et al., 2014], Pascal-VOC [Everingham et al., 2015], etc), and the robotics community (such as BigBIRD [Singh et al., 2014], YCB-Video [Xiang et al., 2018], etc). For grasping, both OpenGrasp [Ulbrich et al., 2011] and VisGraB [Popović et al., 2011] are popular simulation-based benchmarks, whilst the YCB dataset [Calli et al., 2015] focuses on real-world objects. In comparison to these, RLBench allows robotic systems to be evaluated on the complete robotic pipeline, rather than limited to sub-problems such as object detection, state estimation, grasp selection, and planning. Recently there has been an attempt to benchmark the fidelity of simulation environments

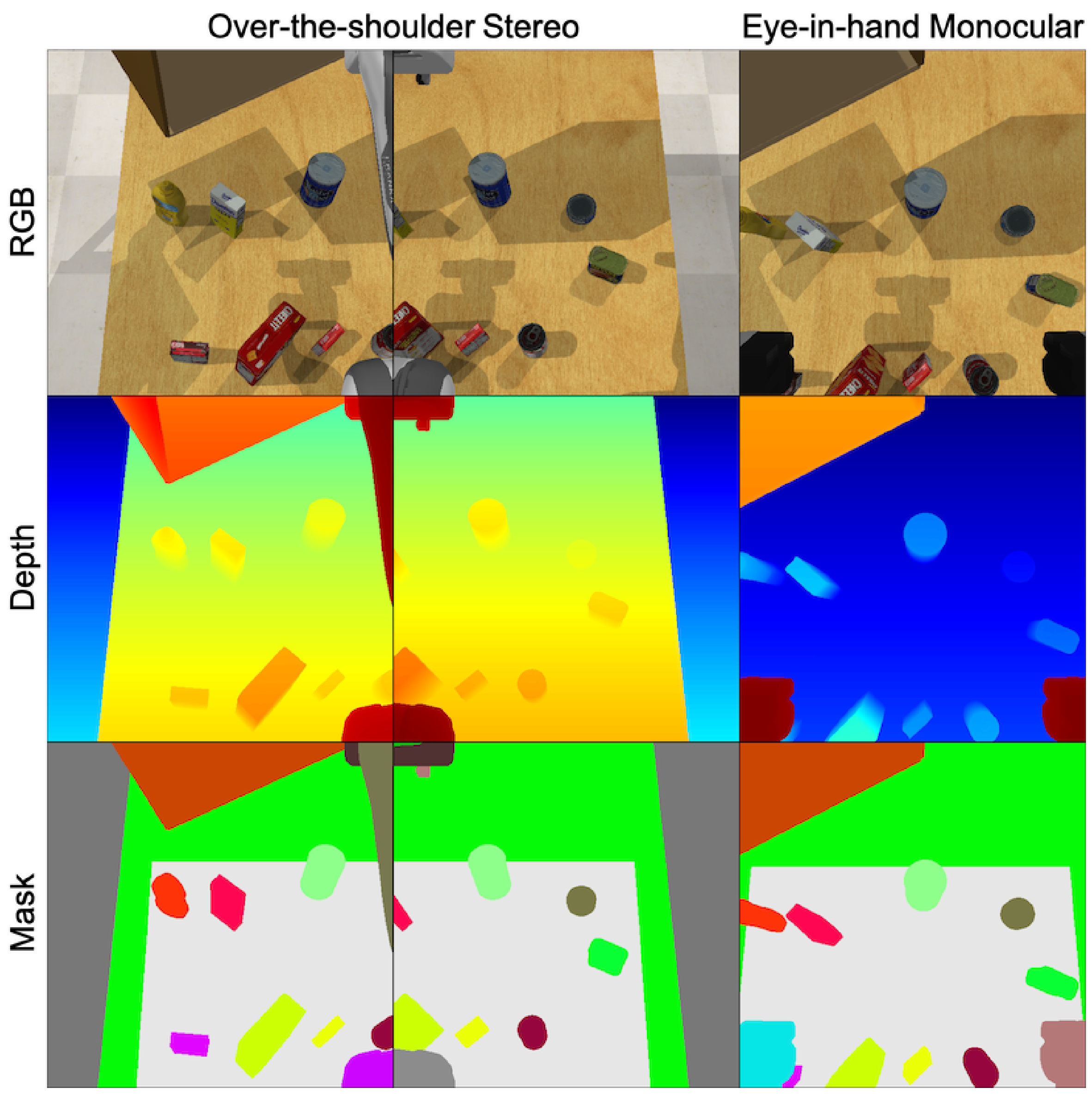


Figure 6.3: A sample of the visual observations given from both the over-the-shoulder stereo and eye-in-hand monocular cameras, which supply rgb, depth, and mask images.

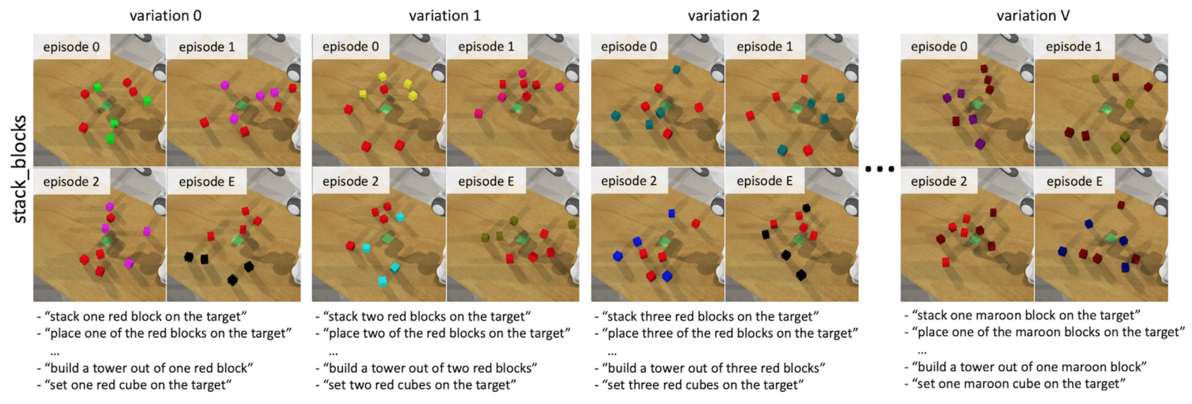
against the real world [Collins et al., 2019].

6.3 Benchmark Properties

When designing RLBench, we have prioritised several key properties:

Diversity Algorithms we develop should be general. In order to effectively learn inter-task relationships, a truly diverse range of tasks is needed to help avoid over-fitting.

Reproducibility Reproducibility is challenging in robotics as each lab has their own robotic setup. Moving to simulation solves this, but at the risk of developing solutions that may not run as well in



- "stack one red block on the target"
- "place one of the red blocks on the target"
...
- "build a tower out of one red block"
- "set one red cube on the target"

- "stack two red blocks on the target"
- "place two of the red blocks on the target"
...
- "build a tower out of two red blocks"
- "set two red cubes on the target"

- "stack three red blocks on the target"
- "place three of the red blocks on the target"
...
- "build a tower out of three red blocks"
- "set three red cubes on the target"

- "stack one maroon block on the target"
- "place one of the maroon blocks on the target"
...
- "build a tower out of one maroon block"
- "set one maroon cube on the target"

Figure 6.4: An example showing the distinction between task, variation, and episode. In this case, the '*stack_blocks*' task has V variations, each with E episodes. Each variation comes with a list of textual descriptions that describes the objective.All of the textual descriptions below each variation in the Figure correspond to the same variation. Across variations, usually target objects or colours are changed, whereas across episodes positions are changed.

the real-world. However, with the rise of deep-learning methods becoming more prominent in robotics, we believe it is important to find the potential and limits of these methods in a controlled, reproducible environment.

Scale Given the amount of data modern machine learning methods need, it is important to not only have a large collection of tasks, but also the ability to produce a large number of demonstrations from these tasks.

Extensibility Following on from the previous point, we hope to continue to grow this repository of tasks. Therefore it is crucial that the task building system is as easy as possible to use. By leveraging the recently released robotics toolkit, PyRep [James et al., 2019a], we are able to make a broad range of tasks in a short amount of time.

Tiered Difficulty Attempting to get robots to do a single task can be challenging let alone expecting them to do numerous tasks. We therefore wanted to have a range of tasks, including both easy tasks, such as reaching, which would be well suited to new and emerging methods, to more challenging, long-time-horizon tasks that can stress-test well known state-of-the-art algorithms in use today.

Realism Although we cannot claim full photorealism in our rendering system, or general realistic physics, we have put substantial effort into high quality components such as using a realistic robot model, graphics with lighting and shadows and a domain randomisation rendering option in order to maximise the potential for research on sim-to-real transfer. Moreover, sensor noise (e.g. depth camera noise, proprioception noise, etc) is also an important part of making the benchmark realistic.

6.4 RLBench

RLBench is an ambitious project which we hope to grow over many years. The benchmark and learning environment is built around a CoppeliaSim/V-REP [Rohmer et al., 2013] and PyRep [James et al., 2019a] interface. PyRep is a toolkit for robot learning research, built on top of CoppeliaSim/V-REP that features a number of improvements, including speed, rendering, and flexible a API for robot control and scene manipulation. Using the combination of these two libraries, we have been able to build this ambitious benchmark, which we now describe in greater detail.

6.4.1 Scene

The CoppeliaSim scene, shown in Figure 6.2, remains constant across all tasks and contains the Franka Emika Panda 7 DoF arm affixed to a wooden table, surrounded by 3 directional lights. As shown in Figure 6.3, visual observations can be perceived from a stereo camera, and a monocular wrist camera, which supply rgb, depth, and segmentation mask data on each frame. In addition to visual observations, robot proprioceptive data can be retrieved, which includes joint angles, velocities, and torques, along with the end-effector pose. To satisfy the realism constraint outlined in the previous section, RLBench comes with the ability to supply a custom noise function for each of the observations; this is vital for ensuring that methods developed in simulation can remain robust to the type of sensor noise we can expect in the real world.

Tasks are loaded into the scene and placed at the centre of the workspace. Each task has a completely sparse reward of $+1$ which is given only on task completion, and 0 otherwise. Users have a wide variety of action spaces at their disposal, which include absolute or delta joint velocities, absolute or delta joint positions, absolute or delta joint torque, absolute or delta end-effector velocities, and finally absolute or delta end-effector poses. Every task starts with the same assumption that no objects are held, therefore, unlike many works in the literature, tasks that involve tools will first need to grasp the object appropriately in order to accomplish the task. Although this makes the environments considerably harder to complete, we believe it is an important assumption to make given that household robots will one day work under such conditions.

Users will interface with the benchmark and learning environment through the *Environment* class. The Environment is the entry point and can spawn child environments, called *TaskEnvironment*, for the tasks you are interested in solving. The environment API, which Figure 6.5 demonstrates, is modelled after a typical agent-environment reinforcement learning setup.

```python
from rlbench.environment import Environment
from rlbench.action_modes import ActionMode
from rlbench.tasks import ReachTarget

DATASET = 'path/to/demo/dataset'

env = Environment(
  DATASET, ActionMode.ABS_JOINT_VELOCITY)
env.launch()

task = env.sample_task()
demos = task.get_demos(2)

agent = Agent()
agent.ingest(demos)

training_steps = 100
episode_length = 100
obs = None
for i in range(training_steps):
  if i % episode_length == 0:
    descriptions, obs = task.reset()
  action = agent.act(obs)
  obs, reward, terminate = task.step(action)
env.shutdown()
```

Figure 6.5: Example usage of the RLBench Environment for training a reinforcement learning agent. When using demonstrations, users can either point to a set of saved demonstrations (as shown here), or alternatively generate demonstrations on the fly.

6.4.2 Tasks, Variations & Episodes

RLBench employs 3 keys terms: *Task*, *Variation*, and *Episode*. Each task consists of one or more variations, and from each variation, an infinite number of episodes can be drawn. Each variation of a task comes with a list of textual descriptions that verbally summarise this variation of the task, which could prove useful for human robot interaction (HRI) and natural language processing (NLP) research. A summary of this can be seen in Figure 6.4. Formally, we define an episode trajectory τ to consist of a series of observations $\mathbf{o}$ and actions $\boldsymbol{a}$: $\tau = [(\mathbf{o}_1, \boldsymbol{a}_1), \ldots, (\mathbf{o}_T, \boldsymbol{a}_T)]$. These episodes are sampled from a variation $\tau \sim \nu$. Finally, we define each task to be a set of variations, $\mathfrak{T} = \{\nu_1, \cdots, \nu_N\}$.

We now motivate the need for the concept of a 'variation' with an example. It is naturally difficult to come up with a precise way to differentiate between tasks given their subjective nature. For example, one could argue that "pick up the apple" and "pick up the banana" are different tasks, whilst one could also equally argue that they are the same "pick up the X" task. We therefore introduce the variation concept, which allows cases like the above to be grouped as very similar tasks. Moreover, given the way the task building tools are designed (discussed in Section 6.4.3), the variation concept allows a convenient way of getting as much from a task definition as possible, given that there is usually only a small amount of additional work needed to generate a large number of variations for a given task.

```
from rlbench.backend.task import Task
from rlbench.backend.conditions import DetectedCondition, GraspedCondition
from pyrep.objects.shape import Shape
from pyrep.objects.proximity_sensor import ProximitySensor

class TakeLidOffSaucepan(Task):

  def init_task(self):
    lid = Shape('saucepan_lid')
    success_detector = ProximitySensor('success')
    self.register_graspable_objects([lid])
    cond_set = [
      GraspedCondition(self.robot.gripper, lid),
      DetectedCondition(lid, success_detector)
    ]
    self.register_success_conditions([cond_set])

  def init_episode(self, index):
    return ['take lid off the saucepan']

  def variation_count(self):
    return 1
```

Figure 6.6: An example of a task python file. When using the task building tool, users are able to simultaneously edit the CoppeliaSim/V-REP scene whilst also changing the various behaviour of a task. In this example, the task is to take a lid off of a saucepan. By interfacing with the scene using PyRep, we register that the episode should terminate and be considered a success only if the saucepan lid is detected by a proximity sensor and that the lid is being held. The backend handles the randomisation of the position of the task at the beginning of each episode.

6.4.3 Task Builder

Two common simulation environments in the literature today are Bullet [Coumans, 2013] and MuJoCo [Todorov et al., 2012b]. However, given that these are physics engines rather than robotics frameworks, it can often be cumbersome to build rich environments and integrate standard robotics tooling such as inverse and forward kinematics, user interfaces, motion libraries, and path planners. Given the scale of RLBench, we needed a tool for designing tasks as easily as possible.

The task building tool is the interface for users who wish to create new tasks to be added to the RLBench task repository. Each task has 2 associated files: a CoppeliaSim/V-REP model file (*.ttm*), which holds all of the scene information and demo waypoints, and a python (*.py*) file, which is responsible for wiring the scene objects to the RLBench backend, applying variations, defining success criteria, and adding other more complex task behaviours. Figure 6.6 shows an example of how simple many tasks files can be.

In order to use the task creator, users must understand how tasks are initialised and placed in the scene. When a user asks for a new task from RLBench, the task is initialised by calling $init\_task()$, and is only called once. Following that, $init\_episode(int\ i)$ is called at the beginning of each episode, and gets passed the variation number, which should be less than or equal to the number of variations for that task (which can be obtained by calling $variation\_count()$). The $init\_episode(int\ i)$ function returns a list of strings which provide descriptions that could be associated with the variation of the task. The list of descriptions are generated by swapping out key words in a set of predefined sentences; for example, a predefined sentence such as ‘put the X block in the Y container’, would allow us to swap out X and Y with different colours depending on the variation number. An analysis of the frequency of words in these descriptions can be seen in top of Figure 6.7.

It is our hope that the suite of tasks offered by RLBench will continue to grow via a community effort. To maintain a high quality of tasks, all tasks submitted to RLBench will be tested using a task validation tool on a continuous integration server. This validation tool ensures that all tasks are well defined and solvable. This is done by sampling a number of episodes across several variations, and ensures that demonstrations can be collected.

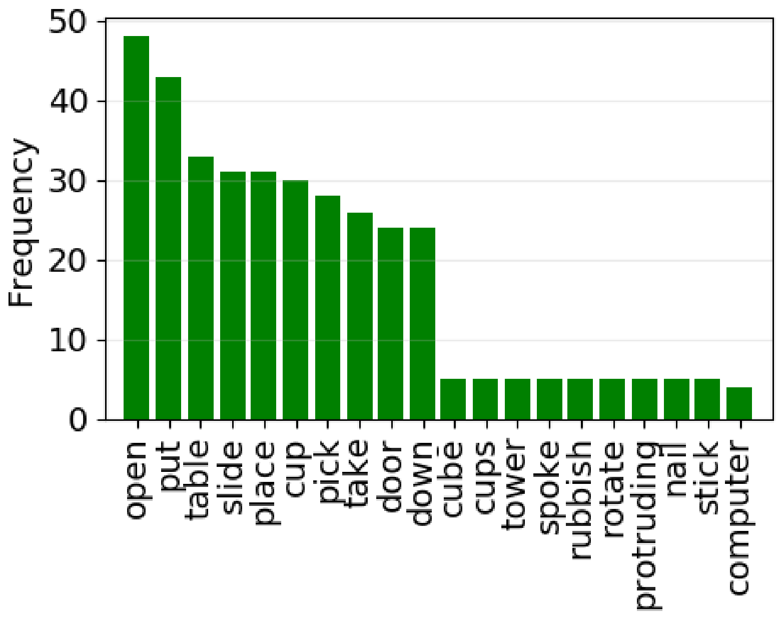
6.4.4 Demonstrations

RLBench, through the task building tool mentioned in Section 6.4.3, provides expert algorithm π^* for each different task and their corresponding variations, allowing for demonstration episodes to be generated. The episodes produced via π^* come from using the Open Motion Planning Library [Sucan et al., 2012].

6.5 The RLBench Few-Shot Challenge

A big gap in the literature today is a means to evaluate and compare few-shot learning methods for robotics. We place particular emphasis on the few-shot regime, because much like humans, robots should have the ability to leverage knowledge from previously learned tasks in order to learn new ones quickly in new and unfamiliar environments. Despite this, most approaches in manipulation have focused on learning a single task, with a limited notion of generalisation, and no way of leveraging the knowledge to learn other tasks more efficiently.

The few pieces of work that perform few-shot learning in robotics [Finn et al., 2017b, James et al., 2018, Yu et al., 2018] focused on a very narrow definition of task and often treat a variation of the same task as another task; for example, placing a peach into a red bowl would be considered a different task to placing an apple into a green bowl. In order to develop truly general algorithms, we feel that it is important to have a diverse range of tasks to train and test on. To that end, we propose the following challenge:



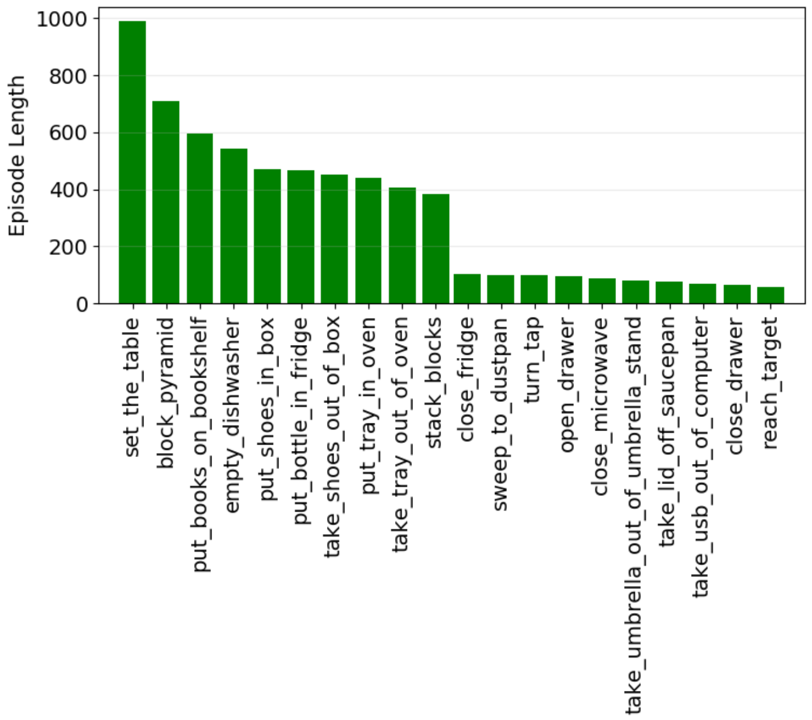


Figure 6.7: **Top** shows the 10 most and 10 least frequent words in the variation descriptions with function words removed, leaving only content words. **Bottom** shows the average length of 5 demonstrations from a sample of 20 tasks (taken from the first variation). The tasks lengths vary from 100 to 1000 timesteps. Longer tasks usually involve many composed sets of actions, for example, the ‘empty_dishwasher’ task involves opening the washer door, sliding out the tray, grasping a plate, and then lifting the plate out of the tray. These long-horizon tasks can facilitate interesting research in reinforcement learning in robotic tasks.

Given N unseen tasks, provide the system with K different demonstrations of each of the N tasks, and then evaluate the systems ability to perform these tasks in new configurations. Specifically, we suggest the following procedure:

- Of the 100 unique tasks, 10% of the tasks have been selected for the test set (meta-test) which span a range of difficulties, while the rest are chosen for training (meta-train). These train-test splits will be made available on the benchmark's webpage.
- The training tasks can be used in any way desired by the user. RLBench supplies a large number of pre-generated demos for each task that can be downloaded, although there is also the option to generate demos on the fly (or for users to create their own).
- During test time, the system is given K demonstrations of the unseen task (K-shot), and then success should be reported on new episodes of that same task. The only information available to the system should be the number of demos N and their corresponding observations. There must be no prior knowledge of the unseen tasks given to the system that are not included in the training tasks. Users report 1-shot, 5-shot, and 20-shot results for their method.

We purposefully call this challenge v 1.0 as we expect the number of tasks to grow considerably over the years; as this happens, we will create newer versions that span a broader range of tasks; therefore, we hope this versioning will ensure results remain meaningful and reproducible as the benchmark grows. State-of-the-art few-shot learning methods such as recurrent methods [Santoro et al., 2016, Duan et al., 2016, Mishra et al., 2017], metric learning methods [Vinyals et al., 2016, Snell et al., 2017], and gradient based methods [Finn et al., 2017a, Rusu et al., 2018] have not been tested on such a grand scale, and we look forward to seeing how they perform on this benchmark.

6.6 Other Applications & Challenges

Further to the few-shot learning challenge highlighted in Section 6.5, we briefly overview other areas of research that could benefit from RLBench.

Reinforcement Learning There is a large body of work in continuous control reinforcement learning that evaluate their algorithms on benchmarks such as OpenAI Gym [Brockman et al., 2016b] or DeepMind Control Suite [Tassa et al., 2018]. Unlike these benchmarks, RLBench has been tailored for visually-guided manipulation, which makes this an ideal platform for evaluating current and future reinforcement learning algorithms on real-world based tasks. Moreover, given the large number of demonstrations provided, it opens up the space to accelerate and facilitate research in bootstrapping reinforcement learning policies with demonstrations in order to reduce sample complexity. In addition, with the provided eye-in-hand camera observations, we open research in partial observability or incremental estimation for continuous control tasks.

Imitation Learning Almost all imitation learning work design their own tasks for evaluating their method, making reproducibility difficult. A set number of demonstrations are shipped with RLBench, but there is also the option in the framework to generate demonstrations on-the-fly, meaning that you cam generate an infinite amount for your imitation learning algorithm.

Sim-to-Real Transfer Recently there has been a large amount of work in learning control policies in simulation and then transferring these to the real world [James et al., 2017a, Peng et al., 2018, Matas et al., 2018, Hwangbo et al., 2019, Bousmalis et al., 2018, James et al., 2019b]. The simulated Franka Panda within RLBench can be easily swapped out, with one line of code, for another arm that researchers may have in their lab; this means that sim-to-real methods could be compared more easily on a standard set of tasks. Moreover, given the task-building tool and demonstration generation that RLbench has to offer, new tasks can easily be designed to demonstrate particular features in novel sim-to-real methods.

Multi-task Learning In contrast to few-shot learning, multi-task learning concerns itself with learning several tasks simultaneously without particularly being expected to generalise to radically different tasks at test time. In this setup, all tasks from both meta-training and meta-testing can be used during training, and then during testing, the system must be able to generalise to unseen examples of those tasks. Given the difficulty of the challenge laid out in Section 6.5, tackling the multi-task problem could provide valuable insights to increasing performance in the few-shot domain.

SLAM and Explicit Model-based Representations There is an increasing acknowledgement from ML researchers that the road to pure end-to-end machine learning of complex, long-term robot behaviour is a very long one, and that there is great value in combining learning with traditionally engineered methods for scene representation [Davison, 2018]. SLAM from moving RGB or depth cameras for instance can build persistent and reliable scene representation at the level of raw dense geometry (e.g. [Newcombe et al., 2011]) or recognised objects (e.g. [McCormac et al., 2018]). While these representations are not perfect, and contain human-designed abstractions which may not be optimal for action, we feel that a crucial and obvious significant ongoing research direction will be on their usefulness for manipulation when combined with task-based action learning. RLBench enables this research with its wide range of tasks, sensor and ground truth support, and could be important in unifying SLAM and manipulation more tightly.

6.7 Conclusion

In this chapter, we have presented RLBench, an attempt to accelerate research in robotic manipulation that can be used in a broad range of robotic related research. We have posed the few-shot learning challenge for manipulation, and have highlighted a number of research areas that could benefit from this large scale benchmark and learning environment.

There are a number of limitations to our benchmark that could be improved over time. An obvious first limitation to our benchmark is the number of tasks; despite having a large number of tasks at launch, we believe this to be an insufficient number in order to accomplish the challenge laid out in Section 6.5. It is our hope that over the next few years, this task repository will grow into the thousands. Another limitation is that object models have been rescaled to unrealistic values. For example, fridges, ovens, dishwashers, and other appliances alike, are much smaller than their real-world counterparts. This was necessary to allow the robot to be able to interact with the object from a variety of starting positions. A trade-off was made between realism, and diversity of starting configurations. Given the choice of robot, we are limited to tasks that can only be accomplished with a single arm. Although RLBench allows for the arm to be swapped with other popular arms, there is currently no support for dual-arm setups. We suggest that this limitation should be fixed by introducing a separate benchmark, specifically aimed at dual-arm systems.

Although benchmarks can drive progress, we must also note the dangers of them. There is a danger that benchmarks can cause researchers to loose sight of the 'bigger picture', and instead focus on developing methods that perform well on a particular benchmark, but do not generalise well to others. Moreover, benchmarks can often promote small, incremental and benchmark-specific improvements to method which in return only give small performance gains. However, when used correctly, benchmarks can rapidly accelerate the speed and quality of research.

Given the scale of this project, we envision that there may be teething problems as people begin using the platform, and so we aim to maintain and continuously improve the benchmark during launch. Further to that, we hope, along with the help of the community, to continuously expand the tasks available for both training and evaluation. We hope RLBench will become a key resource for a broad range of robot manipulation related research, and look forward to seeing what the community achieves with this diverse range of tasks.

CHAPTER 7

Attention Driven Robot Manipulation

Contents

7.1 Introduction

A common theme throughout this thesis has been the incredibly large amount of data that is needed in order to train these data-driven imitation learning or reinforcement learning methods. In particular, continuous-control reinforcement learning (RL) algorithms are are notoriously data hungry, often fail with sparse rewards, and struggle with long-horizon tasks. The algorithms for both discrete and continuous RL are almost always evaluated on benchmarks that give shaped rewards [Brockman et al., 2016b, Tassa et al., 2018], a privilege that is not feasible for training real-world robotic application across a broad range of tasks. Motivated by the observation that humans focus their gaze close to objects being manipulated [Land et al., 1999], we propose an Attention-driven Robotic Manipulation (ARM) algorithm that consists of a series of algorithm-agnostic components, that when combined, results in a method that is able to perform a range of challenging, sparsely-rewarded manipulation tasks.

Our algorithm operates through a pipeline of modules: our novel **Q-attention** module first extracts interesting pixel locations from RGB and point cloud inputs by treating images as an environment, and pixel locations as actions. Using the pixel locations we crop the RGB and point cloud inputs, significantly reducing input size, and feed this to a next-best-pose continuous-control agent that outputs 6D poses, which is trained with our novel **confidence-aware critic**. These goal poses are then used by a control algorithm that continuously outputs motor velocities.

As is common with sparsely-rewarded tasks, we improve initial exploration through the use of demonstrations. However, rather than simply inserting these directly into the replay buffer, we use a **keyframe discovery** strategy that chooses interesting keyframes along demonstration trajectories that is fundamental to training our Q-attention module. Rather than storing the transition from an initial state to a keyframe state, we use our **demo augmentation** method which also stores the transition from intermediate points along a trajectories to the keyframe states; thus greatly increasing the proportion of initial demo transitions in the replay buffer.

All of these improvements result in an algorithm that starkly outperforms other state-of-the-art methods when evaluated on 10 RLBench [James et al., 2020] tasks (Figure 7.1) that range in difficulty. To summarise, we propose the following contributions: **(1)** An off-policy hard attention mechanism that is learned via Q-Learning, rather than the on-policy hard attention and soft attention that is commonly seen in the NLP and vision community; **(2)** A confidence-aware Q function that predicts pixel-wise Q values and confidence values, resulting in improved convergence times; **(3)** A keyframe discovery strategy and demo augmentation method that go hand-in-hand to improve the utilisation of demonstrations in RL.

7.2 Related Work

The use of reinforcement learning (RL) is prevalent in many areas of robotics, including legged robots [Kohl and Stone, 2004, Hwangbo et al., 2019], aerial vehicles [Sadeghi and Levine, 2017], and manipulation tasks, such as pushing [Finn and Levine, 2017b], peg insertion [Levine et al., 2016a, Zeng et al., 2018a, Lee et al., 2019], throwing [Ghadirzadeh et al., 2017, Zeng et al., 2020], ball-in-cup [Kober and Peters, 2009], cloth manipulation [Matas et al., 2018], and grasping [Kalashnikov et al., 2018, James et al., 2019b]. Despite the abundance of work in this area, there has yet to be a general manipulation method that can tackle a range of challenging, sparsely-rewarded tasks without needing access to privileged simulation-only abilities (e.g. reset to demonstrations [Nair et al., 2018], asymmetric actor-critic [Pinto et al., 2018], reward shaping [Rajeswaran et al., 2018], and auxiliary tasks [James et al., 2017a]).

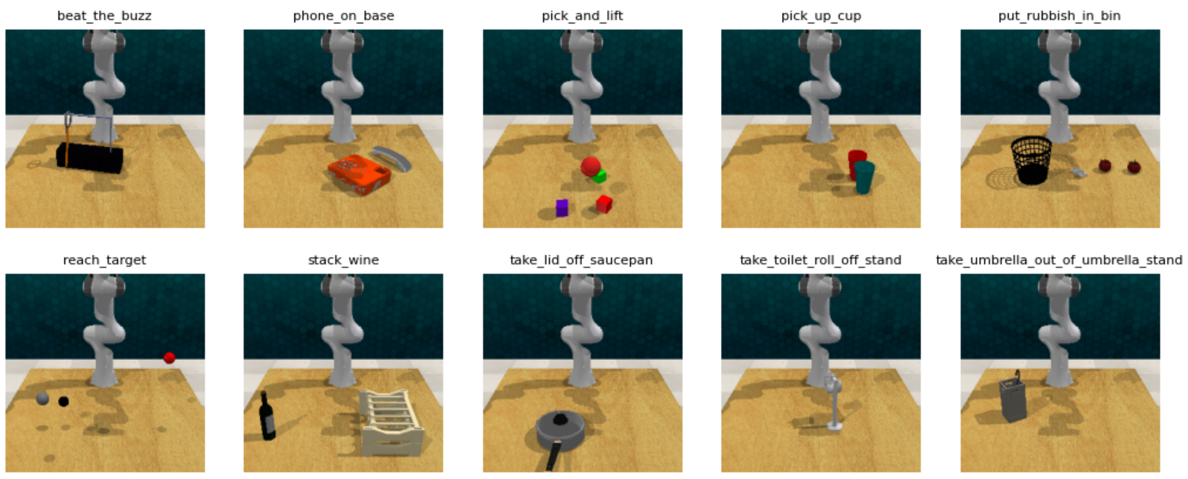


Figure 7.1: The 10 RLBench tasks used for evaluation. Current state-of-the-art reinforcement learning algorithms catastrophically fail on all tasks, whilst our method succeeds within a modest number of steps. Note that the positions of objects are placed randomly at the beginning of each episode.

Crucial to our method is the proposed Q-attention. Soft and hard attention are prominent methods in both natural language processing (NLP) [Bahdanau et al., 2015, Vaswani et al., 2017, Devlin et al., 2018] and computer vision [Xu et al., 2015, Zhang et al., 2019]. Soft attention deterministically multiplies an attention map over the image feature map, whilst hard attention uses the attention map stochastically to sample one or a few features on the feature map (which is optimised by maximising an approximate variational lower bound or equivalently via (on-policy) REINFORCE [Williams, 1992]). Given that we perform non-differentiable cropping, our Q-attention is closest to hard attention, but with the distinction that we learn this in an off-policy way. This is key, as 'traditional' hard attention is unable to be used in an off-policy setting. We therefore see Q-attention as an off-policy hard attention. We elaborate further on these differences in Section 7.4.1.

Our proposed confidence-aware critic (used to train the next-best pose agent) takes its inspiration from the pose estimation community [Wang et al., 2019, Wada et al., 2020]. There exists a small amount of work in estimating uncertainty with Q-learning in discrete domains [Clements et al., 2019, Hoel et al., 2020]; our work uses a continuous Q-function to predict both Q and confidence values for each pixel, which lead to improved stability when training, and is not used during action selection.

Our approach makes use of demonstrations, which has been applied in a number of works [Vecerik et al., 2017, Matas et al., 2018, Kalashnikov et al., 2018, Nair et al., 2018], but while successful, they make limited use of the demonstrations and still can take many samples to converge. Rather than simply inserting these directly into the replay buffer, we instead make sure of our keyframe discovery and demo augmentation to maximise demonstration utility.

7.3 Background

Our Q-attention module builds from Deep Q-learning [Mnih et al., 2015], a method that approximated the value function Q_θ, with a deep convolutional network, whose parameters θ are optimised by sampling mini-batches from a replay buffer $\mathcal{D}$ and using stochastic gradient descent to minimise the loss: $\mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t,\mathbf{s}_{t+1})\sim\mathcal{D}}[(\mathbf{r}+\gamma max_{\mathbf{s}'}Q_{\theta'}(\mathbf{s}_{t+1},\mathbf{s}')-Q_\theta(\mathbf{s}_t,\mathbf{a}_t))^2]$, where $Q_{\theta'}$ is a target network; a periodic copy of the online network Q_θ which is not directly optimised. Our next-best pose agent builds upon SAC [Haarnoja et al., 2018a], however, the agent is compatible with any off-policy, continuous-control RL algorithm. SAC is a maximum entropy RL algorithm that, in addition to maximising the sum of rewards, also maximises the entropy of a policy: $\mathbb{E}_\pi[\sum_t \gamma^t[R(\mathbf{s}_t,\mathbf{a}_t)+\alpha\mathcal{H}(\pi(\cdot|\mathbf{s}_t))]]$, where α is a temperature parameter that determines the relative importance between the entropy and reward. The goal then becomes to maximise a soft Q-function by minimising the following Bellman residual:

$$J_Q(\theta) = \mathop{\mathbb{E}}_{(\mathbf{s}_t,\mathbf{a}_t,\mathbf{s}_{t+1})\sim\mathcal{D}}[((\mathbf{r}+\gamma Q_{\theta'}(\mathbf{s}_{t+1},\pi_\phi(\mathbf{s}_{t+1}))-\alpha\log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t))-Q_\theta(\mathbf{s}_t,\mathbf{a}_t))^2]. \tag{7.1}$$

The policy is updated towards the Boltzmann policy with temperature α, with the Q-function taking the role of (negative) energy. Specifically, the goal is to minimise the Kullback-Leibler divergence between the policy and the Boltzman policy:

$$\pi_{\text{new}} = \arg\min_{\pi'\in\Pi} \mathrm{D_{KL}}\left(\pi'(\cdot|s_t)\,\|\,\frac{\frac{1}{\alpha}\exp\left(Q^{\pi_{\text{old}}}(s_t,\cdot)\right)}{Z^{\pi_{\text{old}}}(s_t)}\right). \tag{7.2}$$

Minimising the expected KL-divergence to learn the policy parameters was shown to be equivalent to maximising the expected value of the soft Q-function:

$$J_\pi(\phi) = \mathop{\mathbb{E}}_{\mathbf{s}_t\sim\mathcal{D}}[\mathop{\mathbb{E}}_{\mathbf{s}\sim\pi_\phi}[\alpha\log(\pi_\phi(\mathbf{a}_t|\mathbf{s}_t))-Q^\pi_\rho(\mathbf{s}_t,\mathbf{a}_t)]]. \tag{7.3}$$

7.4 Method

Our method can be split into a 3-phase pipeline. Phase 1 (Section 7.4.1) consists of a high-level pixel agent that selects areas of interest using our novel Q-attention module. Phase 2 (Section 7.4.2) consists of a next-best pose prediction phase where the pixel location from the previous phase is used to crop the incoming observations and then predict a 6D pose. Finally, phase 3 (Section 7.4.3) is a low-level control agent that accepts the predicted next-best pose and executes a series of actions to reach the given goal. Before training, we fill the replay buffer with demonstrations using our keyframe discovery and

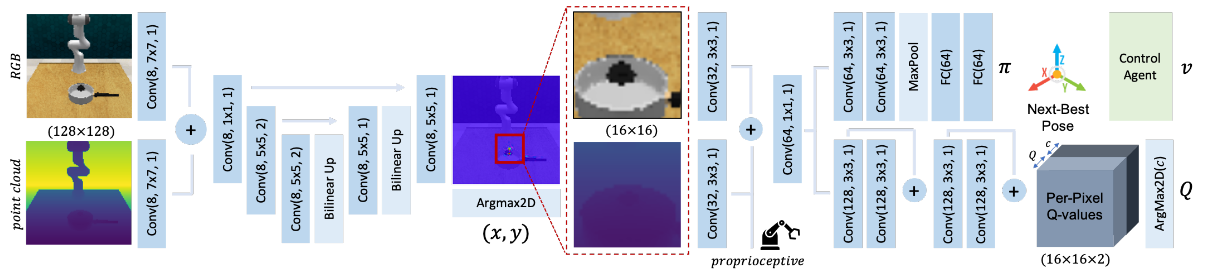


Figure 7.2: Summary and architecture of our method. RGB and organised point cloud crops are made by extracting pixel locations from our Q-attention module. These crops are then fed to a continuous control RL algorithm that suggests next-best poses that is trained with a confidence-aware critic. The next best pose is given to a goal-condition control agent that outputs joint velocities. Conv block represented as *Conv(#channels, filter size, strides)*.

demo augmentation strategy (Section 7.4.4) that significantly improves training speed. The full pipeline is summarised in Figure 7.2 and Algorithm 4.

All experiments are run in RLBench [James et al., 2020], a large-scale benchmark and learning environment for vision-guided manipulation built around CoppeliaSim [Rohmer et al., 2013] and PyRep [James et al., 2019a]. At each time step, we extract an observation from the front-facing camera that consists of an RGB image $\mathbf{b}$ and a depth image $\mathbf{d}$, along with proprioceptive information $\mathbf{z}$ from the arm (consisting of end-effector pose and gripper open/close state). Using known camera intrinsics and extrinsics, we process each depth image to produce a point cloud $\mathbf{p}$ (in world coordinates) projected from the view of the front-facing camera, producing a $(H \times W \times 3)$ 'image'.

7.4.1 Q-attention

Motivated by the role of vision and eye movement in the control of human activities [Land et al., 1999], we propose a Q-attention module that, given RGB and organised point cloud inputs, outputs 2D pixel locations of the next area of interest. With these pixel locations, we crop the RGB and organised point cloud inputs and thus drastically reduce the input size to the next stage of the pipeline. Our Q-attention is explicitly learned via Q-learning, where images are treated as the 'environment', and pixel locations are treated as the 'actions'.

Given our Q-attention function QA_θ, we extract the coordinates of pixels with the highest value:

$$(\mathbf{x}_t, \mathbf{y}_t) = \underset{\mathbf{s}'}{\text{argmax2}D}\, QA_\theta(\mathbf{s}_t, \mathbf{s}'). \tag{7.4}$$

The parameters of the Q-attention are optimised by using stochastic gradient descent to minimise the

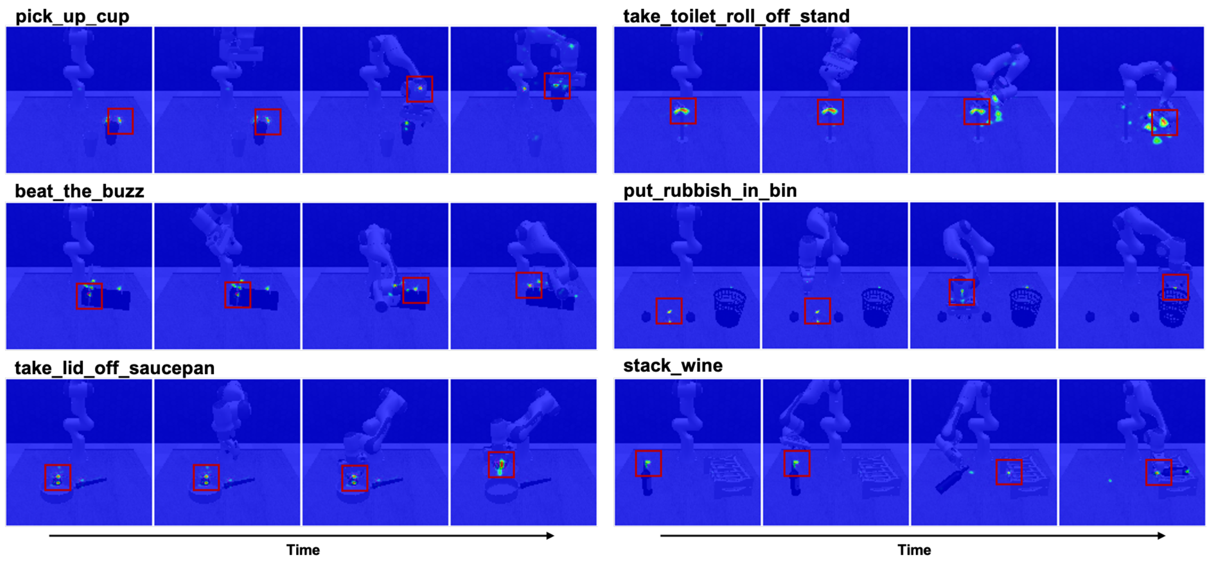


Figure 7.3: Visualising the Q values across 4 different points in time on 6 tasks. At each step, RGB and organised point cloud crops are made by extracting pixel locations that have the highest Q-value; crops are shown via the red squares. We can see that as time progresses, the attention strength shifts depending on progress in the task; e.g. *'stack_wine'* starts with high attention on the bottle, but after grasping, attention shifts to the wine rack.

loss:

$$J_{QA}(\theta) = \mathop{\mathbb{E}}_{(\mathbf{s}_t,\mathbf{a}_t,\mathbf{s}_{t+1})\sim\mathcal{D}}[(\mathbf{r} + \gamma \max_{\mathbf{s}'} 2D\, QA_{\theta'}(\mathbf{s}_{t+1}, \mathbf{s}') - QA_\theta(\mathbf{s}_t, \mathbf{a}_t))^2 + \|QA\|], \tag{7.5}$$

where $\mathbf{s} = (\mathbf{b}, \mathbf{p})$, $QA_{\theta'}$ is the target Q-function, and $\|QA\|$ is an $L2$ loss on the per-pixel output of the Q function (which we call *Q regularisation*); in practice, we found that this leads to increased robustness against the common problem of overestimation of Q values. The Q-attention network follows a light-weight U-Net style architecture [Ronneberger et al., 2015], which is summarised in Figure 7.2. Example per-pixel outputs of the Q-attention are shown in Figure 7.3. With the suggested coordinates from Q-attention, we perform a (16×16) crop on both the (128×128) RGB and organised point cloud data: $\mathbf{b}', \mathbf{p}' = crop(\mathbf{b}, \mathbf{p}, (\mathbf{x}, \mathbf{y}))$.

Notably, there is no explicit reward for choosing a pixel, but instead an implicit reward that comes from the output of the method pipeline as a whole (i.e. the same reward signal is used to train both the Q-attention and the next-best pose agent). This leads to a cyclic dependency between the two agents: the lower-level next-best pose agent relies on receiving good crops from the Q-attention agent, whilst the Q-attention agent needs the next-best pose agent to perform well in order to get its implicit reward. This is where delicate handling of demonstrations is key, which we discuss in Section 7.4.4.

The module shares similar human-inspired motivation to the attention seen in NLP [Bahdanau et al., 2015, Vaswani et al., 2017, Devlin et al., 2018] and computer vision [Xu et al., 2015, Zhang et al., 2019], but differs in its formulation. Soft attention multiplies an attention map over the image feature map, whilst hard attention uses the attention map to sample one or a few features on the feature maps or inputs. Given that we perform non-differentiable cropping, we categorise our Q-attention as hard attention, but with 2 core differences: (1) most importantly, 'traditional' hard attention is optimised (on-policy) by maximising an approximate variational lower bound or equivalently via REINFORCE [Williams, 1992], whereas our Q-attention is trained off-policy; this is crucial because our demonstration data, by definition, is off-policy, and therefore renders existing hard attention approaches unusable for demonstration-driven RL. (2) The output of 'traditional' hard attention carry different semantics: a score function in the case of REINFORCE hard attention, and Q-value (expected cumulative reward of choosing that crop) in the case of Q-attention.

7.4.2 Next-best Pose Agent

Our next-best pose agent accepts cropped RGB $\mathbf{b}'$ and organised point cloud $\mathbf{p}'$ inputs, and outputs a 6D pose. This next-best pose agent is run every time the robot reaches the previously selected pose. We represent the 6D pose via a translation $\mathbf{e} \in \mathcal{R}^3$ and a unit quaternion $\mathbf{q} \in \mathcal{R}^4$, and restrict the w output of $\mathbf{q}$ to a positive number, therefore restricting the network to output unique unit quaternions. The gripper action $\mathbf{h} \in \mathcal{R}^1$ lies between 0 and 1, which is then discretised to a binary open/close value. The combined action therefore is $\mathbf{s} = \{\mathbf{e}, \mathbf{q}, \mathbf{h}\}$.

To train this next-best pose agent, we use a modified version of SAC [Haarnoja et al., 2018a] where we modify the soft Q-function (Equation 7.1) to be a confidence-aware soft Q-function. Recent work in 6D pose estimation [Wang et al., 2019, Wada et al., 2020] has seen the inclusion of a confidence score c with the pose prediction output for each dense-pixel. Inspired by this, we augment our Q function with a per-pixel confidence c_{ij}, where we output a confidence score for each Q-value prediction (resulting in a $(16 \times 16 \times 2)$ output). To achieve this, we weight the per-pixel Bellman loss with the per-pixel confidence, and add a confidence regularisation term:

$$J_{Q^\pi}(\rho) = \mathop{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \sim \mathcal{D}} [((\mathbf{r} + \gamma Q^\pi_{\rho'}(\mathbf{s}_{t+1}, \pi_\phi(\mathbf{s}_{t+1})) - \alpha \log \pi(\mathbf{a}_t|\mathbf{s}_t)) - Q^\pi_\rho(\mathbf{s}_t, \mathbf{a}_t))^2 c - w \log(c)], \quad (7.6)$$

where $\mathbf{s} = (\mathbf{b}', \mathbf{p}', \mathbf{z})$, $\mathbf{b}'$ and $\mathbf{p}'$ are the cropped RGB and point cloud information, and $\mathbf{z}$ is the proprioceptive information from the arm. With this, low confidence will result in a low Bellman error but would incur a high penalty from the second term, and vice versa. We use the Q value that has the highest confidence when training the actor. As an aside, we also tried applying this confidence-aware method to the policy, though empirically we found no improvement. In practice we make use of the clipped

Algorithm 4 ARM

```
Initialise Q-attention networks $QA_{\theta_1}$, $QA_{\theta_2}$, critic networks $Q^{\pi}_{\rho_1}$, $Q^{\pi}_{\rho_2}$, and actor network $\pi_\phi$ with random parameters $\theta_1$, $\theta_2$, $\rho_1$, $\rho_2$, $\phi$
Initialise target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\rho'_1 \leftarrow \rho_1$, $\rho'_2 \leftarrow \rho_2$
Initialise replay buffer $\mathcal{D}$ with demos and apply keyframe selection and demo augmentation
for each iteration do
    for each environment step do
        $(\mathbf{b}_t, \mathbf{p}_t, \mathbf{z}_t) \leftarrow \mathbf{s}_t$
        $(x_t, y_t) \leftarrow \text{argmax}2D_{\mathbf{s}'}\, QA_\theta((\mathbf{b}_t, \mathbf{p}_t), \mathbf{s}')$    ▷ Use Q-attention to get pixel coords
        $\mathbf{b}'_t, \mathbf{p}'_t \leftarrow crop(\mathbf{b}_t, \mathbf{p}_t, (x_t, y_t))$
        $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t | (\mathbf{b}'_t, \mathbf{p}'_t, \mathbf{z}_t))$    ▷ Sample pose from the policy
        while target not reached do
            $v \leftarrow f(\mathbf{s}, \mathbf{a}_t)$    ▷ Get joint velocities from control agent
            $\mathbf{s}_{t+1}, \mathbf{r} \leftarrow env.step(v)$
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}, \mathbf{s}_{t+1}, (x_t, y_t))\}$    ▷ Store the transition in the replay pool
    for each gradient step do
        $\theta_i \leftarrow \theta_i - \lambda_{QA} \hat{\nabla}_{\theta_i} J_{QA}(\theta_i)$ for $i \in \{1, 2\}$    ▷ Update Q-attention parameters
        $\rho_i \leftarrow \rho_i - \lambda_{Q^\pi} \hat{\nabla}_{\rho_i} J_{Q^\pi}(\rho_i)$ for $i \in \{1, 2\}$    ▷ Update critic parameters
        $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$    ▷ Update policy weights
        $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$ for $i \in \{1, 2\}$    ▷ Update Q-attention target network weights
        $\rho'_i \leftarrow \tau\rho_i + (1 - \tau)\rho'_i$ for $i \in \{1, 2\}$    ▷ Update critic target network weights
```

double-Q trick [Fujimoto et al., 2018a], which takes the minimum Q-value between two Q networks, but have omitted in the equations for brevity. Finally, the actor's policy parameters can be optimised by minimising the loss as defined in Equation 7.3.

7.4.3 Control Agent

Given the next-best pose suggestion from the previous stage, we give this to a goal-conditioned control function $f(\mathbf{s}_t, \mathbf{g}_t)$, which given state $\mathbf{s}_t$ and goal $\mathbf{g}_t$, outputs motor velocities that drives the end-effector towards the goal. This function can take on many forms, but two noteworthy solutions would be either motion planning in combination with a feedback-control or a learnable policy trained with imitation/reinforcement learning. Given that the environmental dynamics are limited in the benchmark, we opted for the motion planning solution.

Given the target pose, we perform path planning using the SBL [Sánchez and Latombe, 2003] planner within OMPL [Şucan et al., 2012], and use Reflexxes Motion Library for on-line trajectory generation. If the target pose is out of reach, we terminate the episode and supply a reward of -1. This path planning and trajectory generation is conveniently encapsulated by the *'ABS_EE_POSE_PLAN_WORLD_FRAME'* action mode in RLBench [James et al., 2020].

7.4.4 Keyframe Selection & Demo Augmentation

In this section, we outline how we maximise the utility of given demonstrations in order to complete sparsely reward tasks. We assume to have a teacher policy π^* (e.g. motion planners or human teleoperatives) that can generate trajectories consisting of a series of states and actions: $\tau = [(\mathbf{s}_1, \mathbf{a}_1), \ldots, (\mathbf{s}_T, \mathbf{a}_T)]$. In this case, we assume that the demonstrations come from RLBench [James et al., 2020].

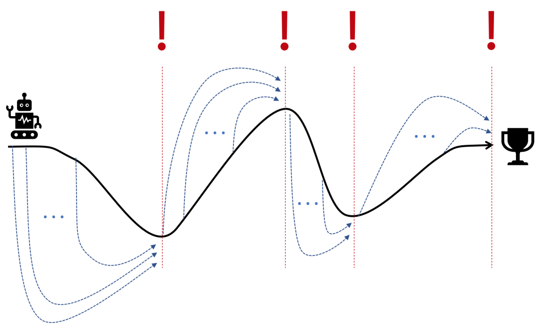


Figure 7.4: Keyframe selection and demo augmentation, where the black line represents a trajectory, '**!**' represents keyframes, and dashed blue lines represent the augmented transitions to the keyframes.

The **keyframe selection** process iterates over each of the demo trajectories τ and runs each of the state-action pairs $(\mathbf{s}, \mathbf{a})$ through a function $K : \mathbb{R}^D \rightarrow \mathbb{B}$ which outputs a Boolean deciding if the given trajectory point should be treated as a keyframe. The keyframe function K could include a number of constraints. In practice we found that performing a disjunction over two simple conditions worked well; these included (1) change in gripper state (a common occurrence when something is grasped or released), and (2) velocities approaching near zero (a common occurrence when entering pre-grasp poses or entering a new phase of a task). It is likely that as tasks get more complex, K will inevitably need to become more sophisticated via learning or simply through more conditions, e.g. sudden changes in direction or joint velocity, large changes in pixel values, etc. Figure 7.5 shows RGB observations from the keyframe selection process from 4 tasks.

At each keyframe, we use the known camera intrinsics and extrinsics to project the end-effector pose at state $\mathbf{s}_{t+1}$ into the image plane of state $\mathbf{s}_t$, giving us pixel locations of the end-effector at the next keyframe. This stage is crucial to breaking the cyclic dependency (mentioned in Section 7.4.1) between the Q-attention and next-best pose agent, as these projected pixel coordinates act as optimal actions for the Q-attention agent.

Using this keyframe selection method, each trajectory results in $N = length(keyframes)$ transitions being stored into the replay buffer. To further increase the utility of demonstrations, we apply **demo augmentation** which stores the transition from an intermediate point along the trajectories to the keyframe states. Formally, for each point $(\mathbf{s}_t, \mathbf{a}_t)$ along the trajectory starting from keyframe k_i, we calculate the transformation of the end-effector pose (taken from $\mathbf{s}_t$) at time step t to the end-effector pose at the time step associated with keyframe k_{i+1}. This transformation can then be used as the action for the next-best pose agent. We repeat this process for every Mth point along the trajectory (which we set to $M = 5$). The keyframe selection and demo augmentation is visualised in Figure 7.4.

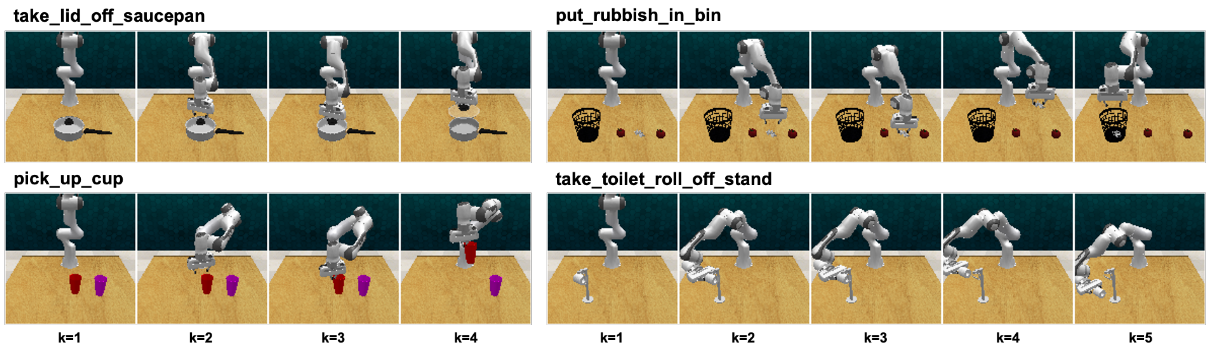


Figure 7.5: Visualising RGB observations of keyframes from the keyframe selection process on 4 tasks. Here k is the keyframe number.

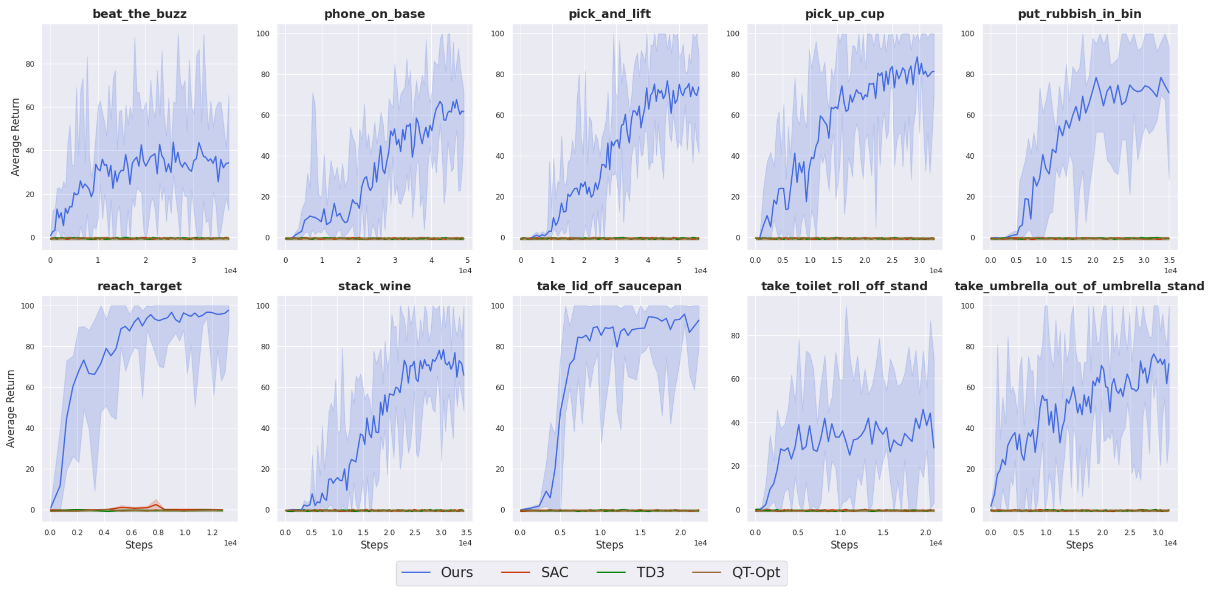


Figure 7.6: Learning curves for 10 RLBench tasks. Methods include Ours (ARM), SAC [Haarnoja et al., 2018a], TD3 [Fujimoto et al., 2018a], and QT-Opt [Kalashnikov et al., 2018]. ARM uses the 3-stage pipeline (Q-attention, next-best pose, and control agent), while baselines use the 2-stage pipeline (next-best pose and control agent). All methods receive 200 demos which are stored in the replay buffer prior to training. Solid lines represent the average evaluation over 5 seeds, where the shaded regions represent the min and max values across those trials.

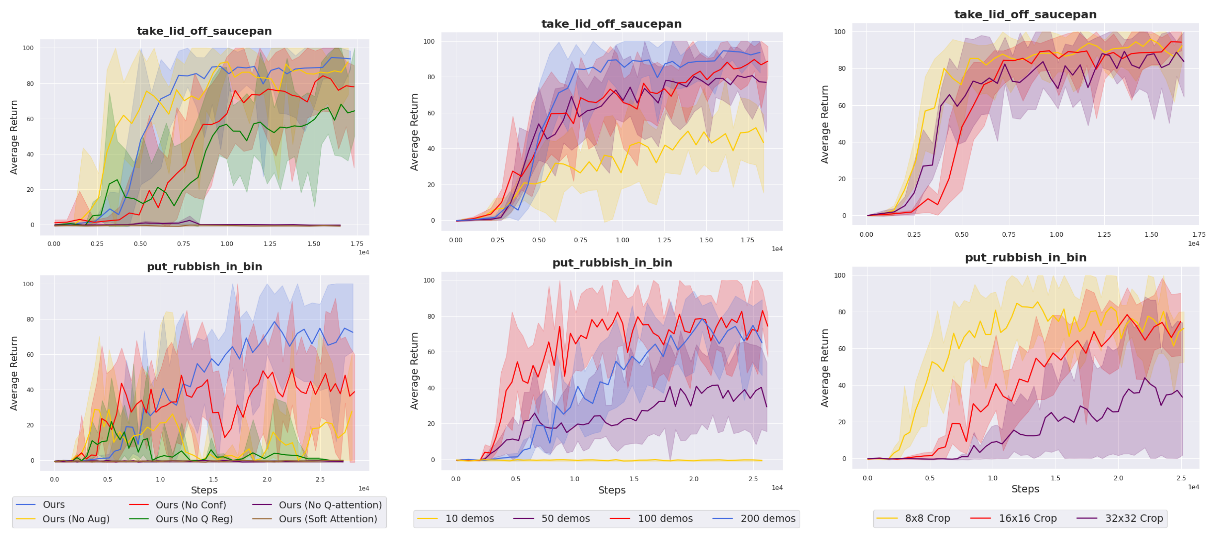
7.5 Results

In this section, we aim to answer the following questions: (1) Are we able to successfully learn across a range of sparsely-rewarded manipulation tasks? (2) Which of our proposed components contribute the most to our success? (3) How sensitive is our method to the number of demonstrations and to the crop size? To answer these, we benchmark our approach using RLBench [James et al., 2020]. Of the 100 tasks, we select 10 (shown in Figure 7.1) that we believe to be achievable from using only the front-facing camera. We leave tasks that require multiple cameras to future work. RLBench was chosen due to its emphasis on vision-based manipulation benchmarking and because it gives access to a wide variety of tasks with demonstrations. Each task has a sparse reward of $+1$ which is given only on task completion, and 0 otherwise.

The first of our questions can be answered by attending to Figure 7.6. All baseline algorithms (SAC, TD3 and QT-Opt) are in their 'vanilla' form, and do not contain any of our proposed contributions: Q-attention, confidence-aware critic, and demo augmentation, but does include the keyframe selection. All methods receive the exact same 200 demonstration sequences, which are loaded into the replay buffer prior to training. The baseline agents are architecturally similar to the next-best pose agent, but with a few differences to account for missing Q-attention (and so receives the full, uncropped RGB and organised point cloud data) and missing confidence-aware critic (and so outputs single Q-values rather than per-pixel values). Specifically, the architecture uses the same RGB and point cloud fusion as shown in Figure 7.2. Feature maps from the shared representation are concatenated with the reshaped proprioceptive input and fed to both the actor and critic. The baseline actor uses 3 convolution layers (64 channels, 3×3 filter size, 2 stride), whose output feature maps are maxpooled and sent through 2 dense layers (64 nodes) and results in an action distribution output. The critic baseline uses 3 residual convolution blocks (128 channels, 3×3 filter size, 2 stride), whose output feature maps are maxpooled and sent through 2 dense layers (64 nodes) and results in a single Q-value output. All methods use the LeakyReLU activation, layer normalisation in the convolution layers, learning rate of 3×10^{-3}, soft target update of $\tau = 5^{-4}$, and a reward scaling of 100. Training and exploration were done asynchronously with a single agent (to emulate a real-world robot training scenario) that would continuously load checkpoints every 100 training steps.

The results in Figure 7.6 show that baseline state-of-the-art methods are unable to accomplish any RLBench tasks, whilst our method is able to accomplish the tasks in small number of environment steps; $5,000$ environment steps equating to about an hour of robot interaction time (meaning *'take_lid_off_saucepan'* being solved in about two hours). We suggest that the reason why our results starkly outperform other state-of-the-art methods is because of two key reasons that go hand-in-hand: (1) Reducing the input dimensionality through Q-attention that immensely reduces the burden on the (often difficult and unstable to train) continuous control algorithm; (2) Combining this with our keyframe selection method that enables the Q-attention network to quickly converge and suggest meaningful points of interest to the next-best pose agent. We wish to stress that perhaps given enough training time some of these baseline methods may eventually start to succeed, however we found no evidence of this.

In Figure 7.7a, we perform an ablation study to evaluate which of the proposed components contribute the most to the success of our method. To perform this ablation, we chose 2 tasks of varying difficulty: *'take_lid_off_saucepan'* and *'put_rubbish_in_bin'*. The ablation clearly shows that the Q-attention (combined with keyframe selection) is crucial to achieving the tasks, whilst the demo augmentation,



(a) Effect of removing components from our method.
(b) Effect of number of demos on performance.
(c) Effect of crop size on performance.

Figure 7.7: Ablation study across the easier *'take_lid_off_saucepan'* task and harder *'put_rubbish_in_bin'* task.

confidence-aware critic, and Q regularisation aid in overall stability and increase final performance. When swapping the Q-attention module with a soft attention [Xu et al., 2015] module, we found that performance was similar to that of the 'vanilla' baselines. This result is unsurprising, as soft attention is implicitly learned (i.e. without an explicit loss), where as our Q-attention is explicitly learned via (off-policy) Q-learning, and so it can make greater use of the highly-informative keyframes given from the keyframe selection process. Note that we cannot compare to 'traditional' hard attention because it requires on-policy training, as explained in Section 7.4.1.

Figure 7.7b shows how robust our method is when varying the number of demonstrations given. The results show that our method performs robustly, even when given 50% fewer demos, however as the task difficulty increase (from *'take_lid_off_saucepan'* to *'put_rubbish_in_bin'*), the harmful effect of having less demonstrations is more severe. Our final set of experiments in Figure 7.7c shows how our method performs across varying crop sizes. As the task difficulty increases, the harmful effect of a larger crop size becomes more prominent; suggesting that one of the key benefits of the Q-attention is to drastically reduce the input size to the next-best pose agent, making the RL optimisation much easier. It is clear that a trade-off must be made between choosing smaller crops to increase training size, and choosing larger crops to incorporate more of the surrounding area. We found that setting the crops at 16×16 across all tasks performed well.

7.6 Conclusion

We have presented our Attention-driven Robotic Manipulation (ARM) algorithm, which is a general manipulation algorithm that can be applied to a range of real-world sparsely-rewarded tasks. We validate our method on 10 RLBench tasks of varying difficulty, and show that many commonly used state-of-the-art methods catastrophically fail. We show that Q-attention (along with the keyframe selection) is key to our success, whilst the confidence-aware critic and demo augmentation contribute to achieving high final performance.

Despite our strong experimental results, there are undoubtedly areas of weakness. The control agent (final agent in the pipeline) uses path planning and on-line trajectory generation, which for these tasks are adequate; however, this would need to be replaced with an alternative agent for tasks that have dynamic environments (e.g. moving target objects, moving obstacles, etc) or complex contact dynamics (e.g. peg-in-hole). We look to future work for swapping this with a goal-conditioned reinforcement learning policy, or similar. Another weakness is that we only evaluate on tasks that can be done with the front-facing camera; however we are keen to explore many of the other tasks RLBench has to offer by adapting the method to accommodate multiple camera inputs in future work.

CHAPTER 8

Conclusions and Future Work

In this thesis, we have presented a number of contributions towards finding where on the spectrum, from pipelined to fully end-to-end, is best suited for creating a general manipulation system. That is, a system that could eventually be usable in highly dynamic and chaotic household environments. Our investigation began in Chapter 3, at the far end of the spectrum, where we explored learning an end-to-end controller for a cube-in-basket task in simulation. Transfer to the real world was made possible by using domain randomisation, such that through a large amount of variability in the appearance of the world, the model was able to generalise to real world environments. This work was the first of its kind by showing that we could transfer end-to-end controllers for multi-stage tasks, and has since been applied in many works [Zhang et al., 2017, Yan et al., 2017, Zhu et al., 2018, Hämäläinen et al., 2019, Iqbal et al., 2020]. Moreover, we showed that the same trained controller worked in novel situations, including those with dynamic lighting conditions, distractor objects, and moving objects, including the basket itself.

In Chapter 4, we push the boundaries of domain randomisation by asking if sim-to-real transfer was possible with deformable objects, not just rigid ones. The answer turns out to be yes, which is fortunate as deformable objects usually have large configuration spaces, meaning that solutions using traditional modelling approaches require significant engineering work. The chapter explored the use of a combination of state-of-the-art deep reinforcement learning algorithms to solve the problem of manipulating cloth in a variety of tasks, including: folding a towel up to a mark, folding a face towel diagonally, and draping a piece of cloth over a hanger. We showed that we are able to fully train in simulation with domain randomisation, and then successfully deploy in the real world without having seen any real deformable objects.

With the knowledge that end-to-end methods work well for individual tasks, Chapter 5 explored how best to learn new tasks from one or a few demonstrations with Task-Embedded Control Networks. This was the first time in the thesis that we moved from a ‘monolith’ style, single network end-to-end system to a 2-part moduled system, which consisted of the task-embedding module and the control module. Although being separate networks with their own responsibility and losses, it remained crucial to train these in an end-to-end manner (i.e. propagating gradients from the control network to the embedding network). This resulted in a far richer embedding space that was suited not only to separate tasks, but also to produce embeddings that were interpretable by the control network. The chapter also explored a way of endowing robots with the ability of imitating humans from third person. To do this, we extended Task-Embedded Control Networks to infer control polices by observing videos of humans performing the desired task. What was unique about this work was its use of domain randomisation in an application that has not been seen before: sim-to-real transfer on humans. Upon evaluating our approach on pushing and placing tasks in both simulation and in the real world, we showed that in comparison to a system that was trained on real-world data we were able to achieve similar results by utilising only simulation data.

Despite achieving high success on one-shot imitation learning with Task-Embedded Control Networks, the system had not been tested with limited variation across tasks. The core reason for this was the lack of a suitable benchmark or standard set of tasks for the few-shot manipulation community. It was this that motivated RLBench, our new benchmark and learning-environment for robot learning, presented in Chapter 6. The benchmark featured 100 completely unique, hand-designed tasks, with the aim to facilitate research in a number of vision-guided manipulation research areas, including: reinforcement learning, imitation learning, multi-task learning, geometric computer vision, and few-shot learning. RLBench now has an active GitHub community, and has been used to benchmark several recent papers [Chen et al., 2020, Kim et al., 2020, Choi et al., 2020, Lee et al., 2020].

What makes RLBench so appealing is how challenging it is for end-to-end methods. In fact, upon evaluating many image-based state-of-the-art reinforcement learning algorithms on RLBench, it was clear that all end-to-end approaches failed catastrophically in these complex, sparsely-rewarded tasks. This motivated the need for a more structured approach to the problem, and drove our research direction further along the spectrum, to produce a 3-stage manipulation pipeline. Our Attention-driven Robotic Manipulation (ARM) algorithm, which we presented in Chapter 7, was a general manipulation algorithm that could be applied to a range of real-world sparse-rewarded tasks without any prior task knowledge. ARM split the complex task of manipulation into a 3 stage pipeline: (1) a Q-attention agent extracts interesting pixel locations from RGB and point cloud inputs, (2) a next-best pose agent that accepts crops from the Q-attention agent and outputs poses, and (3) a control agent that takes the goal pose and outputs joint actions.

There is a clear evolution through the thesis, where in the initial chapters, we scrapped the ‘traditional’ manipulation pipeline, and started afresh with an end-to-end approach. As the chapters progressed, we gradually modularised, and finally ended with ARM, where the individual modules bare little resemblance to the 'traditional' ones described in the opening chapters. What's important to note here is that it no longer makes sense to classify ARM as an end-to-end method, as there are distinct modules that are not jointly optimised via gradients. However, this also cannot be classed as a 'traditional' pipelined method because action is still tightly coupled with input observation through the shared reward, and there is no object detection, pose estimation, grasp planning, or motion planning stages. With that in mind, we conclude by proposing to classify this paradigm as a **Tightly-coupled Manipulation Pipeline (TMP)**. Rather than learning all modules implicitly in one large, end-to-end network or conversely, having individual, pre-defined modules that are developed independently, TMPs suggest taking the best of both world by tightly coupling actions to observations, whilst still maintaining structure via an undefined number of learned modules, which do not have to bare any resemblance to the modules seen in 'traditional' systems. This tight coupling of TMPs can be achieved either implicitly through gradients flowing through each of the modules, such as in Chapter 5, or implicitly via a shared reward, such as in Chapter 7.

In the future, we would like to explore alternative sim-to-real methods; in work external to this thesis, we explored how domain adaptation lead to a higher zero-shot performance in the real world when compared to domain randomisation [James et al., 2019b]. This begs the question as to what other methods would perform better than domain randomisation? Not only this, but what methods are best equipped to be fine-tuned on additional real-world data in order to refine a simulation-trained controller. It has been suggested that learning a policy directly on domain randomisation can act as a very powerful pre-training regime, where the network is forced to learn a very general feature extractor that can be easily jointly fine-tuned to a new environment [James et al., 2019b]. If so, then what other methods exist to achieve a similar effect? Gradient based meta learning is a powerful method for learning networks that be quickly adapted at test time, and so perhaps a combination of sim-to-real and meta learning could be trained in simulation for fast adaptation to real-world data at test time.

Given the potential use cases of RLBench, we envision that it will continue to grow over the years, and so we look to maintain and continuously improve the benchmark during this time. A natural extension to this in future work, could be the introduction of a mobile base. This would not only make the action and state space more complex, but would dramatically increase the range of possible tasks that could be included. A limitation of RLBench is that the breadth of tasks are limited to those that can be done on the table-top workspace in the scene; moving to a mobile base would open up the workspace to a much larger range of tasks. Moreover, this would bring in additional challenges, such as localisation, mapping, and long-horizon planning, to name but a few.

We are excited by the idea of combing our Attention-driven Robotic Manipulation (ARM) system with few-shot learning. A future direction could be to combine ARM with Task-Embedded Control Networks (TECNets) to solve the meta learning challenge posed in RLBench. ARM gives us a system than can be trained quickly on individual tasks, whereas TECNets gives us a system that can adapt to new tasks given a few demonstrations. However, given the limited variation of tasks that TECNets has been evaluated on, and given that the diversity of tasks in RLBench is so great, it is unreasonable to expect that TECNets will work without modification. It is therefore reasonable to accept that there will need to be some degree of optimisation at test time; finding a method that can do this effectively in a small amount of time will be the challenge. Finally, and possibly most importantly, we are excited to explore other Tightly-coupled Manipulation Pipelines that go beyond ARM.

Recall that in Figure 1.2 of Chapter 1, we showed where our presented methods fell on the spectrum, from fully end-to-end to pipelined. It is clear that the focus of this thesis has mostly been on end-to-end rather than on pipelined methods. Given the time constraints of the PhD, developing novel pipelined methods was not possible. However in hindsight, existing pipelines could have been implemented to act as comparison baselines to the end-to-end approaches put forward. For example, in Chapter 3, object detection and pose estimation networks could have been trained for various cubes and baskets and then subsequently used for grasp planning and placement. Another example from Chapter 4, could have been to estimate the state of the cloth and preform model-predictive control for action selection. These pipeline baselines would have allowed the thesis to cover the entire spectrum, rather than only part of it.

To conclude this thesis, we discuss our general outlook on the future of robot manipulation given our experience throughout the PhD. We split our outlook into 4 key areas: mobile manipulation, dexterous multi-finger manipulation, machine learning community shift, and Tightly-coupled Manipulation Pipelines.

The large majority of work in robot manipulation focuses on a robot fixed to a surface operating within a constrained workspace. Although one can do a lot in this scenario (which RLBench attests), it does however greatly limit the number of tasks that can be achieved in a household environment. Not only that, but it ignores many of the other problems within the area of robotics, such as navigation, spatial awareness, spacial memory, and long term planning, to name but a few. By only evaluating systems in these non-mobile settings, we may miss out on important insights and modes of failure that are only present in a mobile domain.

We see dexterous multi-finger manipulation as another key exciting future area. This is already an active area of research, however it often seems detached from the larger problem at hand. Many papers look at dexterous in-hand manipulation in isolation, where the hand is not attached to a manipulator. In the rare cases when a hand is attached to a manipulator, vision is rarely used. Given the physical limit to what a robot can accomplish with parallel-jaw grippers, we therefore speculate that soon we will have to look at other end-effectors, and multi-finger hands seem like an attractive prospect.

There is evidence that Embodied AI (the study of intelligent systems with a physical or virtual embodiment, e.g. robots and egocentric personal assistants) has been gaining traction in the machine learning and computer vision communities; this is clear by the increasing number of Embodied AI simulations and workshops at conferences such as CVPR, ICCV, and NeurIPS, to name but a few. We therefore predict that this move from static datasets and supervised learning, to the more dynamic and challenging domain of Embodied AI, will continue to gain popularity. This would mean an influx of new research working at the intersection of robotics and machine learning, potentially leading to a large surge of work.

Finally, as this thesis suggests, we see tightly-coupled manipulation pipelines playing a big role in future manipulation systems. As our final chapter shows, taking the best of both end-to-end and pipelines by tightly coupling actions to observations and maintaining structure leads to a powerful system. As the tasks get more complex and have increasingly longer time horizons, it seems unlikely that a monolith end-to-end system will be able to navigate, plan, perform scene understanding, have spacial memory, perform complex control, and perform many other important processes. On the other end of the spectrum, it also seems unlikely that a pre-programmed pipeline will be general enough to operate on a plethora of tasks without having to be re-engineered each time. It therefore seems that TMP systems are correctly placed to be an ideal direction for future research.

Bibliography

[Abbeel et al., 2006] Abbeel, P., Coates, A., Quigley, M., and Ng, A. Y. (2006). An Application of Reinforcement Learning to Aerobatic Helicopter Flight. *International Conference on Neural Information Processing Systems*. 63

[Abbeel and Ng, 2004] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. *International Conference on Machine learning*. 23, 78

[Abdolmaleki et al., 2018] Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018). Maximum a posteriori policy optimisation. *arXiv preprint arXiv:1806.06920*. 23

[Ahmadzadeh et al., 2013] Ahmadzadeh, S. R., Kormushev, P., and Caldwell, D. G. (2013). Visuospatial skill learning for object reconfiguration tasks. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 685–691. IEEE. 90

[Akgun et al., 2012] Akgun, B., Cakmak, M., Yoo, J. W., and Thomaz, A. L. (2012). Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM. 23

[Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., Mcgrew, B., Tobin, J., Abbeel, P., and Openai, W. Z. (2017). Hindsight Experience Replay. *Neural Information Processing Systems Conference*. 64

[Argall et al., 2009] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems*. 77

[Ba et al., 2016] Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*. 83, 95

[Bahdanau et al., 2015] Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *Intl. Conference on Learning Representations*. 119, 122

[Balaguer and Carpin, 2011] Balaguer, B. and Carpin, S. (2011). Combining imitation and reinforcement learning to fold deformable planar objects. *International Conference on Intelligent Robots and Systems*. 63

[Barth-Maro et al., 2018] Barth-Maro, G., Hoffma, M. W., Budden, D., Dabney, W., Horgan, D., Tb, D., Muldal, A., Heess, N., Lillicrap, T., and London (2018). Distributed Distributional Deterministic Policy Gradients. *International Conference on Learning Representations*. 64, 65

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Van Gool, L. (2008). Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359. 16

[Bellet et al., 2013] Bellet, A., Habrard, A., and Sebban, M. (2013). A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*. 78

[Bersch et al., 2011] Bersch, C., Pitzer, B., and Kammel, S. (2011). Bimanual robotic cloth manipulation for laundry folding. *International Conference on Intelligent Robots and Systems*. 62

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics. 17

[Bohg et al., 2014] Bohg, J., Morales, A., Asfour, T., and Kragic, D. (2014). Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics*, 30(2):289–309. 18

[Bonardi et al., 2020] Bonardi, A., James, S., and Davison, A. J. (2020). Learning one-shot imitation from humans without humans. *IEEE Robotics and Automation Letters*. 13, 25

[Bousmalis et al., 2018] Bousmalis, K., Irpan, A., Wohlhart, P., Bai, Y., Kelcey, M., Kalakrishnan, M., Downs, L., Ibarz, J., Pastor, P., Konolige, K., et al. (2018). Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *IEEE Intl. Conference on Robotics and Automation*. 90, 93, 97, 114

[Brachmann et al., 2014] Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6d object pose estimation using 3d object coordinates. In *European Conference on Computer Vision*, pages 536–551. Springer. 17

[Brockman et al., 2016a] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016a). OpenAI Gym. *GitHub repository*. 69

[Brockman et al., 2016b] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016b). Openai gym. *arXiv preprint arXiv:1606.01540*. 101, 103, 104, 113, 117

[Brook et al., 2011] Brook, P., Ciocarlie, M., and Hsiao, K. (2011). Collaborative grasp planning with multiple object representations. *IEEE Intl. Conference on Robotics and Automation*. 18

[Calinon and Billard, 2006] Calinon, S. and Billard, A. (2006). Teaching a humanoid robot to recognize and reproduce social cues. In *ROMAN 2006-The 15th IEEE International Symposium on Robot and Human Interactive Communication*, pages 346–351. IEEE. 23

[Calinon et al., 2009] Calinon, S., Evrard, P., Gribovskaya, E., Billard, A., and Kheddar, A. (2009). Learning collaborative manipulation tasks by demonstration using a haptic interface. In *2009 International Conference on Advanced Robotics*, pages 1–6. IEEE. 23

[Calli et al., 2015] Calli, B., Walsman, A., Singh, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. (2015). Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *Robotics & Automation Magazine*. 21, 105

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., et al. (2015). Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*. 86

[Chatfield et al., 2014] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. (2014). Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531*. 40

[Chebotar et al., 2017] Chebotar, Y., Hausman, K., Zhang, M., Sukhatme, G., Schaal, S., and Levine, S. (2017). Combining model-based and model-free updates for trajectory-centric reinforcement learning. *arXiv preprint arXiv:1703.03078*. 22

[Chebotar et al., 2016] Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S. (2016). Path integral guided policy search. *arXiv preprint arXiv:1610.00529*. 22

[Chen et al., 2020] Chen, B., Sax, A., Lewis, G., Armeni, I., Savarese, S., Zamir, A., Malik, J., and Pinto, L. (2020). Robust policies via mid-level visual representations: An experimental study in manipulation and navigation. *arXiv preprint arXiv:2011.06698*. 132

[Choi et al., 2020] Choi, J., Kim, H., Son, Y., Park, C.-W., and Park, J. H. (2020). Robotic behavioral cloning through task building. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 1279–1281. IEEE. 132

[Choi et al., 2018] Choi, P. J., Oskouian, R. J., and Tubbs, R. S. (2018). Telesurgery: past, present, and future. *Cureus*, 10(5). 15

[Christiano et al., 2016] Christiano, P., Shah, Z., Mordatch, I., Schneider, J., Blackwell, T., Tobin, J., Abbeel, P., and Zaremba, W. (2016). Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*. 50

[Ciocarlie et al., 2014] Ciocarlie, M., Hsiao, K., Jones, E. G., Chitta, S., Rusu, R. B., and Şucan, I. A. (2014). Towards reliable grasping and manipulation in household environments. In *Experimental Robotics*, pages 241–252. Springer. 18

[Clements et al., 2019] Clements, W. R., Robaglia, B.-M., Van Delft, B., Slaoui, R. B., and Toth, S. (2019). Estimating risk and uncertainty in deep reinforcement learning. *arXiv preprint arXiv:1905.09638*. 119

[Clevert et al., 2016] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). *International Conference on Learning Representation*. 83, 95

[Collet et al., 2011] Collet, A., Martinez, M., and Srinivasa, S. S. (2011). The moped framework: Object recognition and pose estimation for manipulation. *The International Journal of Robotics Research*, 30(10):1284–1306. 16

[Collins et al., 2019] Collins, J., McVicar, J., Wedlock, D., Brown, R., Howard, D., and Leitner, J. (2019). Benchmarking simulated robotic manipulation through a real world dataset. *RA-L*. 106

[Correll et al., 2016] Correll, N., Bekris, K. E., Berenson, D., Brock, O., Causo, A., Hauser, K., Okada, K., Rodriguez, A., Romano, J. M., and Wurman, P. R. (2016). Analysis and observations from the first amazon picking challenge. *IEEE Transactions on Automation Science and Engineering*, 15(1):172–188. 20

[Coumans, 2013] Coumans, E. (2013). Bullet real-time physics simulation. https://pybullet.org/wordpress/. 110

[Coumans, 2015] Coumans, E. (2015). Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, SIGGRAPH '15, New York, NY, USA. ACM. 31

[Coumans and Bai, 2016] Coumans, E. and Bai, Y. (2016). PyBullet, a Python module for physics simulation for games, robotics and machine learning. *GitHub repository*. 67, 69

[Cusumano-Towner et al., 2011] Cusumano-Towner, M., Singh, A., Miller, S., O'Brien, J. F., and Abbeel, P. (2011). Bringing clothing into desired configurations with limited perception. *International Conference on Robotics and Automation*. 62, 63

[Dasari et al., 2019] Dasari, S., Ebert, F., Tian, S., Nair, S., Bucher, B., Schmeckpeper, K., Singh, S., Levine, S., and Finn, C. (2019). Robonet: Large-scale multi-robot learning. *arXiv preprint arXiv:1910.11215*. 104

[Davison, 2018] Davison, A. J. (2018). FutureMapping: The computational structure of Spatial AI systems. *arXiv preprint arXiv:arXiv:1803.11288*. 114

[DeepMind, 2019] DeepMind (2019). Alphastar: Mastering the real-time strategy game starcraft ii. `https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii`. 103

[Devin et al., 2017] Devin, C., Gupta, A., Darrell, T., Abbeel, P., and Levine, S. (2017). Learning modular neural network policies for multi-task and multi-robot transfer. *Robotics and Automation (ICRA)*. 50, 101

[Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. 119, 122

[Devol, 1961] Devol, J. G. C. (1961). Programmed article transfer. US Patent 2,988,237. 15

[Dhariwal et al., 2017] Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2017). Openai baselines. *GitHub repository*. 67

[Dillmann, 2004] Dillmann, R. (2004). Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems*, 47(2-3):109–116. 23

[Dilokthanakul et al., 2017] Dilokthanakul, N., Kaplanis, C., Pawlowski, N., and Shanahan, M. (2017). Feature control as intrinsic motivation for hierarchical reinforcement learning. *arXiv preprint arXiv:1705.06769*. 48

[Drost et al., 2010] Drost, B., Ulrich, M., Navab, N., and Ilic, S. (2010). Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 998–1005. Ieee. 17

[Duan et al., 2017] Duan, Y., Andrychowicz, M., Stadie, B., Ho, J., Schneider, J., Sutskever, I., Abbeel, P., and Zaremba, W. (2017). One-shot imitation learning. *arXiv preprint arXiv:1703.07326*. 49

[Duan et al., 2016] Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*. 113

[E. Rohmer, 2013] E. Rohmer, S. P. N. Singh, M. F. (2013). V-rep: a versatile and scalable robot simulation framework. *International Conference on Intelligent Robots and Systems (IROS)*. 51, 94

[Ekvall and Kragic, 2004] Ekvall, S. and Kragic, D. (2004). Interactive grasp learning based on human demonstration. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3519–3524. IEEE. 23

[Elbanhawi and Simic, 2014] Elbanhawi, M. and Simic, M. (2014). Sampling-based robot motion planning: A review. *Ieee access*, 2:56–77. 19

[Eppner et al., 2015] Eppner, C., Deimel, R., Álvarez-Ruiz, J., Maertens, M., and Brock, O. (2015). Exploitation of environmental constraints in human and robotic grasping. *The International Journal of Robotics Research*, page 0278364914559753. 19

[Eppner et al., 2016] Eppner, C., Höfer, S., Jonschkowski, R., Martín-Martín, R., Sieverling, A., Wall, V., and Brock, O. (2016). Lessons from the amazon picking challenge: Four aspects of building robotic systems. In *Robotics: Science and Systems*. 104

[Everingham et al., 2015] Everingham, M., Eslami, S. A., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*. 20, 105

[Fikes et al., 1972] Fikes, R. E., Hart, P. E., and Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial intelligence*, 3:251–288. 15

[Finn et al., 2017a] Finn, C., Abbeel, P., and Levine, S. (2017a). Model-agnostic meta-learning for fast adaptation of deep networks. In *Intl. Conference on Machine Learning*. 76, 78, 79, 113

[Finn and Levine, 2017a] Finn, C. and Levine, S. (2017a). Deep visual foresight for planning robot motion. *International Conference on Robotics and Automation (ICRA)*. 49

[Finn and Levine, 2017b] Finn, C. and Levine, S. (2017b). Deep visual foresight for planning robot motion. In *IEEE Intl. Conference on Robotics and Automation*, pages 2786–2793. IEEE. 118

[Finn et al., 2016] Finn, C., Levine, S., and Abbeel, P. (2016). Guided cost learning: Deep inverse optimal control via policy optimization. *International Conference on Machine Learning*. 23, 78

[Finn et al., 2017b] Finn, C., Yu, T., Zhang, T., Abbeel, P., and Levine, S. (2017b). One-shot visual imitation learning via meta-learning. *Conference on Robot Learning*. 76, 77, 78, 79, 82, 83, 84, 85, 96, 99, 101, 102, 111

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395. 16

[Frome et al., 2013] Frome, A., Corrado, G. S., Shlens, J., Bengio, S., Dean, J., Mikolov, T., et al. (2013). Devise: A deep visual-semantic embedding model. *Advances in Neural Information Processing Systems*. 79

[Fujimoto et al., 2018a] Fujimoto, S., van Hoof, H., and Meger, D. (2018a). Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*. 23, 103, 124, 126

[Fujimoto et al., 2018b] Fujimoto, S., van Hoof, H., and Meger, D. (2018b). Addressing Function Approximation Error in Actor-Critic Methods. *CoRR*. 64, 66

[Gao et al., 2016] Gao, Y., Chang, H. J., and Demiris, Y. (2016). Iterative path optimisation for personalised dressing assistance using vision and force information. *International Conference on Intelligent Robots and Systems*. 61

[Ghadirzadeh et al., 2017] Ghadirzadeh, A., Maki, A., Kragic, D., and Björkman, M. (2017). Deep predictive policy training using reinforcement learning. In *IEEE Intl. Conference on Intelligent Robots and Systems*, pages 2351–2358. IEEE. 118

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1440–1448. 18

[Girshick et al., 2014] Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587. 17

[Goertz, 1952a] Goertz, R. C. (1952a). A force-reflecting positional servomechanism. *Nucleonics*, 10(11):43–45. 15

[Goertz, 1952b] Goertz, R. C. (1952b). Fundamentals of general-purpose remote manipulators. *Nucleonics*, 10(11):36–42. 15, 20

[Goertz, 1954] Goertz, R. C. (1954). Mechanical master-slave manipulator. *Nucleonics (US) Ceased publication*, 12. 15

[Goertz, 1963] Goertz, R. C. (1963). Manipulators used for handling radioactive materials. *Human factors in technology*, pages 425–443. 15

[Goertz and Thompson, 1954] Goertz, R. C. and Thompson, W. M. (1954). Electronically controlled manipulator. *Nucleonics (US) Ceased publication*, 12. 15

[Gu et al., 2016] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2016). Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. *International Conference on Robotics and Automation*. 63

[Gu et al., 2017] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *IEEE Intl. Conference on Robotics and Automation*. 49

[Gupta et al., 2017] Gupta, A., Devin, C., Liu, Y., Abbeel, P., and Levine, S. (2017). Learning invariant feature spaces to transfer skills with reinforcement learning. *International Conference on Learning Representations (ICLR)*. 50

[Haarnoja et al., 2018a] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*. 23, 103, 120, 123, 126

[Haarnoja et al., 2018b] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018b). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*. 23

[Hagbi et al., 2011] Hagbi, N., Bergig, O., El-Sana, J., and Billinghurst, M. (2011). Shape recognition and pose estimation for mobile augmented reality. *IEEE transactions on visualization and computer graphics*, 17(10):1369–1379. 16

[Hämäläinen et al., 2019] Hämäläinen, A., Arndt, K., Ghadirzadeh, A., and Kyrki, V. (2019). Affordance learning for end-to-end visuomotor robot control. *arXiv preprint arXiv:1903.04053*. 131

[Harlow, 1949] Harlow, H. F. (1949). The formation of learning sets. *Psychological review*. 75

[Hausman et al., 2017] Hausman, K., Chebotar, Y., Schaal, S., Sukhatme, G., and Lim, J. (2017). Multimodal imitation learning from unstructured demonstrations using generative adversarial nets. *Conference on Neural Information Processing Systems (NIPS)*. 49

[Hausman et al., 2018] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. (2018). Learning an embedding space for transferable robot skills. *Intl. Conference on Learning Representations*. 78, 101

[Havoutis and Calinon, 2019] Havoutis, I. and Calinon, S. (2019). Learning from demonstration for semi-autonomous teleoperation. *Autonomous Robots*, 43(3):713–726. 15

[He et al., 2016a] He, F. S., Liu, Y., Schwing, A. G., and Peng, J. (2016a). Learning to play in a day: Faster deep reinforcement learning by optimality tightening. *arXiv preprint arXiv:1611.01606*. 23

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2961–2969. 18

[He et al., 2016b] He, K., Zhang, X., Ren, S., and Sun, J. (2016b). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778. 17

[Hernandez et al., 2016] Hernandez, C., Bharatheesha, M., Ko, W., Gaiser, H., Tan, J., van Deurzen, K., de Vries, M., Van Mil, B., van Egmond, J., Burger, R., et al. (2016). Team delft's robot winner of the amazon picking challenge 2016. *arXiv preprint arXiv:1610.05514*. 18, 19

[Hessel et al., 2017] Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G., and Silver, D. (2017). Rainbow: Combining improvements in deep reinforcement learning. *CoRR*. 67

[Higgins et al., 2018] Higgins, I., Pal, A., Rusu, A. A., Matthey, L., Burgess, C. P., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. (2018). Darla: Improving zero-shot transfer in reinforcement learning. *Intl. Conference on Machine Learning*. 47

[Hinterstoisser et al., 2012a] Hinterstoisser, S., Cagniart, C., Ilic, S., Sturm, P., Navab, N., Fua, P., and Lepetit, V. (2012a). Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888. 16

[Hinterstoisser et al., 2011] Hinterstoisser, S., Holzer, S., Cagniart, C., Ilic, S., Konolige, K., Navab, N., and Lepetit, V. (2011). Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 858–865. IEEE. 16, 18

[Hinterstoisser et al., 2012b] Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012b). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer. 16

[Hoel et al., 2020] Hoel, C.-J., Wolff, K., and Laine, L. (2020). Tactical decision-making in autonomous driving by reinforcement learning with uncertainty estimation. *arXiv preprint arXiv:2004.10439*. 119

[Hogan, 1984] Hogan, N. (1984). Impedance control: An approach to manipulation. In *1984 American Control Conference*, pages 304–313. 20

[Holzer et al., 2012] Holzer, S., Shotton, J., and Kohli, P. (2012). Learning to efficiently detect repeatable interest points in depth data. In *European Conference on Computer Vision*, pages 200–213. Springer. 16

[Hsu et al., 1997] Hsu, D., Latombe, J.-C., and Motwani, R. (1997). Path planning in expansive configuration spaces. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2719–2726. IEEE. 19

[Huttenlocher et al., 1993] Huttenlocher, D. P., Klanderman, G. A., and Rucklidge, W. J. (1993). Comparing images using the hausdorff distance. *IEEE Transactions on pattern analysis and machine intelligence*, 15(9):850–863. 16

[Hwangbo et al., 2019] Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872. 114, 118

[Insights, 2020] Insights, F. B. (2020). Market research report. `https://www.fortunebusinessinsights.com/industry-reports/industrial-robots-market-100360`. 15

[Iqbal et al., 2020] Iqbal, S., Tremblay, J., Campbell, A., Leung, K., To, T., Cheng, J., Leitch, E., McKay, D., and Birchfield, S. (2020). Toward sim-to-real directional semantic grasping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7247–7253. IEEE. 131

[Jaderberg et al., 2016] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *International Conference for Learning Representations (ICLR)*. 48

[Jain et al., 1996] Jain, A. K., Mao, J., and Mohiuddin, K. (1996). Artificial neural networks: A tutorial. *Computer*, pages 31–44. 34

[James et al., 2018] James, S., Bloesch, M., and Davison, A. J. (2018). Task-embedded control networks for few-shot imitation learning. *Conference on Robot Learning*. 13, 25, 97, 101, 111

[James and Davison, 2021] James, S. and Davison, A. J. (2021). Attention-driven robotic manipulation. *Under Review*. 14, 24

[James et al., 2017a] James, S., Davison, A. J., and Johns, E. (2017a). Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. *Conference on Robot Learning*. 12, 23, 25, 31, 76, 78, 86, 90, 93, 94, 97, 114, 118

[James et al., 2017b] James, S., Davison, A. J., and Johns, E. (2017b). Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task. *Conference on Robot Learning*. 62, 64

[James et al., 2019a] James, S., Freese, M., and Davison, A. J. (2019a). Pyrep: Bringing v-rep to deep robot learning. *arXiv preprint arXiv:1906.11176*. 25, 94, 102, 104, 107, 108, 121

[James and Johns, 2016a] James, S. and Johns, E. (2016a). 3d simulation for robot arm control with deep q-learning. *NIPS Workshop (Deep Learning for Action and Interaction)*. 23, 47, 49, 56, 76, 101

[James and Johns, 2016b] James, S. and Johns, E. (2016b). 3D Simulation for Robot Arm Control with Deep Q-Learning. *Neural Information Processing Systems Conference Workshop*. 64

[James et al., 2020] James, S., Ma, Z., Rovick Arrojo, D., and Davison, A. J. (2020). RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*. 13, 24, 118, 121, 124, 125, 126

[James et al., 2019b] James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019b). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *IEEE Conference on Computer Vision and Pattern Recognition*. 26, 31, 90, 93, 97, 114, 118, 133

[Jiang et al., 2011] Jiang, Y., Moseson, S., and Saxena, A. (2011). Efficient grasping from rgbd images: Learning using a new rectangle representation. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3304–3311. IEEE. 18

[Johns et al., 2016] Johns, E., Leutenegger, S., and Davison, A. J. (2016). Deep learning a grasp function for grasping under gripper pose uncertainty. *Intelligent Robots and Systems (IROS)*. 50

[Kaelbling et al., 1996] Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, pages 237–285. 42

[Kahn et al., 2016] Kahn, G., Zhang, T., Levine, S., and Abbeel, P. (2016). Plato: Policy learning using adaptive trajectory optimization. *arXiv preprint arXiv:1603.00622*. 21

[Kalashnikov et al., 2018] Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*. 23, 101, 118, 119, 126

[Kappler et al., 2015] Kappler, D., Bohg, J., and Schaal, S. (2015). Leveraging big data for grasp planning. *IEEE Intl. Conference on Robotics and Automation*. 18

[Karaman and Frazzoli, 2011] Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894. 19

[Kavraki et al., 1996] Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580. 19

[Kehoe et al., 2013] Kehoe, B., Matsukawa, A., Candido, S., Kuffner, J., and Goldberg, K. (2013). Cloud-based robot grasping with the google object recognition engine. In *IEEE Intl. Conference on Robotics and Automation*. 18

[Kim et al., 2020] Kim, S., Jang, I., Kim, H., Park, C.-W., and Park, J. H. (2020). Learning robot manipulation based on modular reward shaping. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 883–886. IEEE. 132

[Kingma and Ba, 2015] Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference for Learning Representations (ICLR)*. 54, 83, 95

[Kober et al., 2013] Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274. 23

[Kober and Peters, 2009] Kober, J. and Peters, J. R. (2009). Policy search for motor primitives in robotics. In *Advances in Neural Information Processing Systems*, pages 849–856. 118

[Koch et al., 2015] Koch, G., Zemel, R., and Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. *ICML Deep Learning Workshop*. 78

[Kohl and Stone, 2004] Kohl, N. and Stone, P. (2004). Machine learning for fast quadrupedal locomotion. In *Association for the Advancement of Artificial Intelligence*, volume 4, pages 611–616. 118

[Kormushev et al., 2011] Kormushev, P., Calinon, S., and Caldwell, D. G. (2011). Imitation learning of positional and force skills demonstrated via kinesthetic teaching and haptic input. *Advanced Robotics*, 25(5):581–603. 23

[Kormushev et al., 2013] Kormushev, P., Calinon, S., and Caldwell, D. G. (2013). Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2(3):122–148. 23

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 17

[Kruger et al., 2010] Kruger, V., Herzog, D. L., Baby, S., Ude, A., and Kragic, D. (2010). Learning actions from observations. *IEEE robotics & automation magazine*, 17(2):30–43. 23

[Kuffner and LaValle, 2000] Kuffner, J. J. and LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE. 19

[Kulis et al., 2012] Kulis, B. et al. (2012). Metric learning: A survey. *Foundations and Trends in Machine Learning*. 78

[Lai et al., 2013] Lai, K., Bo, L., Ren, X., and Fox, D. (2013). Rgb-d object recognition: Features, algorithms, and a large scale benchmark. In *Consumer Depth Cameras for Computer Vision*, pages 167–192. Springer. 20

[Land et al., 1999] Land, M., Mennie, N., and Rusted, J. (1999). The roles of vision and eye movements in the control of activities of daily living. *Perception*, 28(11):1311–1328. 117, 121

[Laskey et al., 2017] Laskey, M., Powers, C., Joshi, R., Poursohi, A., and Goldberg, K. (2017). Learning Robust Bed Making using Deep Imitation Learning with DART. *CoRR*. 61, 63

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 17, 37

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2. 17

[Lee et al., 2015a] Lee, A. X., Gupta, A., Lu, H., Levine, S., and Abbeel, P. (2015a). Learning from multiple demonstrations using trajectory-aware non-rigid registration with applications to deformable object manipulation. *Intelligent Robots and Systems (IROS)*. 49

[Lee et al., 2015b] Lee, A. X., Gupta, A., Lu, H., Levine, S., and Abbeel, P. (2015b). Learning from Multiple Demonstrations using Trajectory-Aware Non-Rigid Registration with Applications to Deformable Object Manipulation. *International Conference on Intelligent Robots and Systems*. 63, 69

[Lee et al., 2020] Lee, D., Kim, H., Kim, S., Park, C.-W., and Park, J. H. (2020). Learning control policy with previous experiences from robot simulator. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 863–865. IEEE. 132

[Lee and Ryoo, 2017] Lee, J. and Ryoo, M. S. (2017). Learning robot activities from first-person human videos using convolutional future regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–2. 24

[Lee et al., 2013] Lee, K., Su, Y., Kim, T.-K., and Demiris, Y. (2013). A syntactic approach to robot imitation learning using probabilistic activity grammars. *Robotics and Autonomous Systems*, 61(12):1323–1334. 24, 90

[Lee et al., 2019] Lee, M. A., Zhu, Y., Srinivasan, K., Shah, P., Savarese, S., Fei-Fei, L., Garg, A., and Bohg, J. (2019). Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *IEEE Intl. Conference on Robotics and Automation*, pages 8943–8950. IEEE. 118

[Leitner et al., 2016] Leitner, J., Tow, A. W., Dean, J. E., Suenderhauf, N., Durham, J. W., Cooper, M., Eich, M., Lehnert, C., Mangels, R., McCool, C., et al. (2016). The acrv picking benchmark (apb): A robotic shelf picking benchmark to foster reproducible research. *arXiv preprint arXiv:1609.05258*. 19, 21

[Leitner et al., 2017] Leitner, J., Tow, A. W., Sünderhauf, N., Dean, J. E., Durham, J. W., Cooper, M., Eich, M., Lehnert, C., Mangels, R., McCool, C., et al. (2017). The acrv picking benchmark: A robotic shelf picking benchmark to foster reproducible research. In *IEEE Intl. Conference on Robotics and Automation*. 104

[Lemke et al., 2015] Lemke, C., Budka, M., and Gabrys, B. (2015). Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*. 77

[Lenz et al., 2015] Lenz, I., Lee, H., and Saxena, A. (2015). Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724. 18

[Lever, 2014] Lever, G. (2014). Deterministic policy gradient algorithms. *International Conference on Machine Learning (ICML)*. 23

[Levine and Abbeel, 2014] Levine, S. and Abbeel, P. (2014). Learning neural network policies with guided policy search under unknown dynamics. *Advances in Neural Information Processing Systems (NIPS)*. 21, 49

[Levine et al., 2016a] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016a). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*. 21, 47, 49, 76, 90, 118

[Levine and Koltun, 2013] Levine, S. and Koltun, V. (2013). Guided policy search. *Intl. Conference on Machine Learning*. 21, 49

[Levine et al., 2016b] Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016b). Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *International Symposium on Experimental Robotics*. 18

[Levine et al., 2016c] Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016c). Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *International Symposium on Experimental Robotics (ISER)*. 19, 49

[Li et al., 2015] Li, Y., Yue, Y., Xu, D., Grinspun, E., and Allen, P. (2015). Folding Deformable Objects using Predictive Simulation and Trajectory Optimization. *International Conference on Intelligent Robots and Systems*. 62, 63

[Lillicrap et al., 2015a] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015a). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. 21, 23, 103

[Lillicrap et al., 2015b] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015b). Continuous control with deep reinforcement learning. *CoRR*. 62, 63, 64

[Lin, 1992] Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321. 22

[Lin et al., 2014] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European Conference on Computer Vision*. 20, 105

[Lisle, 2020] Lisle, D. (2020). Making safe: The dirty history of a bomb disposal robot. *Security Dialogue*, 51(2-3):174–193. 15

[Lowe, 2001] Lowe, D. G. (2001). Local feature view clustering for 3d object recognition. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE. 16

[Lowe, 2004] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110. 16

[Lozano-Pérez and Wesley, 1979] Lozano-Pérez, T. and Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570. 19

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*. 87

[Mahler et al., 2017] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *Robotics: Science and Systems*. 18

[Maitin-Shepard et al., 2010] Maitin-Shepard, J., Cusumano-Towner, M., Lei, J., and Abbeel, P. (2010). Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. *International Conference on Robotics and Automation*. 62

[Mandlekar et al., 2018] Mandlekar, A., Zhu, Y., Garg, A., Booher, J., Spero, M., Tung, A., Gao, J., Emmons, J., Gupta, A., Orbay, E., et al. (2018). Roboturk: A crowdsourcing platform for robotic skill learning through imitation. *arXiv preprint arXiv:1811.02790*. 104

[Matas et al., 2018] Matas, J., James, S., and Davison, A. J. (2018). Sim-to-real reinforcement learning for deformable object manipulation. *Conference on Robot Learning*. 12, 23, 25, 31, 86, 90, 93, 114, 118, 119

[McCormac et al., 2018] McCormac, J., Clark, R., Bloesch, M., Davison, A. J., and Leutenegger, S. (2018). Fusion++:volumetric object-level slam. In *3DV*. 114

[Memmesheimer et al., 2019] Memmesheimer, R., Mykhalchyshyna, I., Seib, V., and Paulus, D. (2019). Simitate: A hybrid imitation learning benchmark. *arXiv preprint arXiv:1905.06002*. 104

[Miller and Allen, 2004] Miller, A. T. and Allen, P. K. (2004). Graspit! a versatile simulator for robotic grasping. *IEEE Robotics & Automation Magazine*, 11(4):110–122. 19

[Miller et al., 2014] Miller, S., van den Berg, J., Fritz, M., Darrell, T., Goldberg, K., and Abbeel, P. (2014). A Geometric Approach to Robotic Laundry Folding. *Household Service Robotics*. 61

[Mishra et al., 2017] Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. (2017). A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*. 113

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*. 22, 23, 44, 49, 103, 120

[Montgomery et al., 2016] Montgomery, W., Ajay, A., Finn, C., Abbeel, P., and Levine, S. (2016). Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. *International Conference on Robotics and Automation (ICRA)*. 21, 22, 47, 49

[Montgomery and Levine, 2016] Montgomery, W. H. and Levine, S. (2016). Guided policy search via approximate mirror descent. *Advances in Neural Information Processing Systems (NIPS)*. 21, 22, 47, 49

[Morrison et al., 2018] Morrison, D., Tow, A. W., McTaggart, M., Smith, R., Kelly-Boxall, N., Wade-McCue, S., Erskine, J., Grinover, R., Gurman, A., Hunn, T., et al. (2018). Cartman: The low-cost cartesian manipulator that won the amazon robotics challenge. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7757–7764. IEEE. 30, 104

[Nair et al., 2017] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Overcoming Exploration in Reinforcement Learning with Demonstrations. *International Conference on Robotics and Automation*. 64, 66

[Nair et al., 2018] Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *IEEE Intl. Conference on Robotics and Automation*, pages 6292–6299. IEEE. 23, 118, 119

[Newcombe et al., 2011] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *IEEE Intl. Symposium on Mixed and Augmented Reality*. 114

[Ng et al., 2000] Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. *International Conference on Machine Learning*. 23, 78

[Nilsson, 1984] Nilsson, N. J. (1984). Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA. 15

[Nister and Stewenius, 2006] Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, volume 2, pages 2161–2168. Ieee. 16

[OpenAI, 2018] OpenAI (2018). Openai five. `https://blog.openai.com/openai-five/`. 103

[Osawa et al., 2007] Osawa, F., Seki, H., and Kamiy, Y. (2007). Unfolding of Massive Laundry and Classification Types by Dual Manipulator. *Journal of Advanced Computational Intelligence and Intelligent Informatics*. 62

[Osband et al., 2016] Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016). Deep exploration via bootstrapped dqn. In *Advances In Neural Information Processing Systems*, pages 4026–4034. 23

[O'Shea and Nash, 2015] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*. 40

[Pastor et al., 2011] Pastor, P., Righetti, L., Kalakrishnan, M., and Schaal, S. (2011). Online movement adaptation based on previous sensor experiences. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 365–371. IEEE. 23

[Peng et al., 2018] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE Intl. Conference on Robotics and Automation*. 114

[Peng et al., 2017] Peng, X. B., Berseth, G., and Kangkang, Y. (2017). DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Transactions on Graphics*. 63

[Perlin, 1985] Perlin, K. (1985). An image synthesizer. *ACM SIGGRAPH Computer Graphics*. 67

[Perlin, 2002] Perlin, K. (2002). Improving noise. *ACM Transactions on Graphics (TOG)*. 52

[Peters and Schaal, 2008] Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*. 63

[Philbin et al., 2007] Philbin, J., Chum, O., Isard, M., Sivic, J., and Zisserman, A. (2007). Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE. 16

[Pinto et al., 2017] Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. (2017). Asymmetric Actor Critic for Image-Based Robot Learning. *Robotics: Science and Systems*. 64, 66

[Pinto et al., 2018] Pinto, L., Andrychowicz, M., Welinder, P., Zaremba, W., and Abbeel, P. (2018). Asymmetric actor critic for image-based robot learning. *Robotics: Science and Systems*. 118

[Pinto and Gupta, 2016a] Pinto, L. and Gupta, A. (2016a). Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours. *IEEE Intl. Conference on Robotics and Automation*. 18

[Pinto and Gupta, 2016b] Pinto, L. and Gupta, A. (2016b). Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *International Conference on Robotics and Automation (ICRA)*. 19, 49

[Pomerleau, 1989] Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. *Advances in Neural Information Processing Systems*. 23, 77

[Popov et al., 2017] Popov, I., Heess, N., Lillicrap, T., Hafner, R., Barth-Maron, G., Vecerik, M., Lampe, T., Tassa, Y., Erez, T., and Riedmiller, M. (2017). Data-efficient deep reinforcement learning for dexterous manipulation. *arXiv preprint arXiv:1704.03073*. 47, 49

[Popović et al., 2011] Popović, M., Kootstra, G., Jørgensen, J. A., Kragic, D., and Krüger, N. (2011). Grasping unknown objects using an early cognitive vision system for general scene understanding. In *IEEE Intl. Conference on Intelligent Robots and Systems*. 21, 105

[Pratt and Manzo, 2013] Pratt, G. and Manzo, J. (2013). The darpa robotics challenge. *IEEE Robotics Automation Magazine*, 20(2):10–12. 20

[Prattichizzo and Trinkle, 2008] Prattichizzo, D. and Trinkle, J. C. (2008). Grasping. In *Springer handbook of robotics*, pages 671–700. Springer. 18

[Quillen et al., 2018] Quillen, D., Jang, E., Nachum, O., Finn, C., Ibarz, J., and Levine, S. (2018). Deep Reinforcement Learning for Vision-Based Robotic Grasping: A Simulated Comparative Evaluation of Off-Policy Methods. *CoRR*. 62

[Rajeswaran et al., 2018] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *Robotics: Science and Systems*. 118

[Ralovich et al., 2014] Ralovich, K., John, M., Camus, E., Navab, N., and Heimann, T. (2014). 6dof catheter detection, application to intracardiac echocardiography. In *MICCAI (2)*, pages 635–642. 16

[Ramirez-Amaro et al., 2017] Ramirez-Amaro, K., Beetz, M., and Cheng, G. (2017). Transferring skills to humanoid robots by extracting semantic representations from observations of human activities. *Artificial Intelligence*, 247:95–118. 24

[Ravi and Larochelle, 2017] Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. *International Conference on Learning Representations*. 78

[Redmon and Angelova, 2015] Redmon, J. and Angelova, A. (2015). Real-time grasp detection using convolutional neural networks. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1316–1322. IEEE. 18

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99. 18

[Rennie et al., 2016] Rennie, C., Shome, R., Bekris, K. E., and De Souza, A. F. (2016). A dataset for improved rgbd-based object detection and pose estimation for warehouse pick-and-place. *IEEE Robotics and Automation Letters*, 1(2):1179–1185. 20

[Rios-Cabrera and Tuytelaars, 2013] Rios-Cabrera, R. and Tuytelaars, T. (2013). Discriminatively trained templates for 3d object detection: A real time scalable approach. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2048–2055. 16

[Rodriguez et al., 2012] Rodriguez, A., Mason, M. T., and Ferry, S. (2012). From caging to grasping. *The International Journal of Robotics Research*, 31(7):886–900. 18

[Rohmer et al., 2013] Rohmer, E., Singh, S. P., and Freese, M. (2013). V-rep: A versatile and scalable robot simulation framework. *IROS*. 31, 104, 108, 121

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer. 86, 122

[Ross et al., 2011] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. *International Conference on Artificial Intelligence and Statistics*. 23, 77

[Rosten et al., 2010] Rosten, E., Porter, R., and Drummond, T. (2010). Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):105–119. 16

[Rothfuss et al., 2018] Rothfuss, J., Ferreira, F., Aksoy, E. E., Zhou, Y., and Asfour, T. (2018). Deep episodic memory: Encoding, recalling, and predicting episodic experiences for robot action execution. *IEEE Robotics and Automation Letters*, 3(4):4007–4014. 24

[Russakovsky et al., 2015a] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015a). ImageNet Large Scale Visual Recognition Challenge. *IJCV*. 105

[Russakovsky et al., 2015b] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015b). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252. 17, 20

[Rusu et al., 2018] Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. (2018). Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*. 113

[Rusu et al., 2017a] Rusu, A. A., Vecerik, M., Rothorl, T., Heess, N., Pascanu, R., and Hadsell, R. (2017a). Sim-to-real robot learning from pixels with progressive nets. *Conference on Robot Learning (CoRL)*. 49

[Rusu et al., 2017b] Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2017b). Sim-to-Real Robot Learning from Pixels with Progressive Nets. *Conference on Robot Learning*. 64

[Sadeghi and Levine, 2016] Sadeghi, F. and Levine, S. (2016). (cad)2rl: Real single-image flight without a single real image. *Robotics: Science and Systems Conference (R:SS)*. 48, 50

[Sadeghi and Levine, 2017] Sadeghi, F. and Levine, S. (2017). Cad2rl: Real single-image flight without a single real image. *Robotics: Science and Systems*. 93, 118

[Sánchez and Latombe, 2003] Sánchez, G. and Latombe, J.-C. (2003). A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *Robotics research*, pages 403–417. Springer. 124

[Santoro et al., 2016] Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., and Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In *Intl. Conference on Machine Learning*. 78, 113

[Schaal, 1999] Schaal, S. (1999). Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*. 77

[Schaul et al., 2015a] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015a). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*. 23

[Schaul et al., 2015b] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized Experience Replay. *CoRR*. 64, 65

[Schulman et al., 2013] Schulman, J., Ho, J., Lee, C., and Abbeel, P. (2013). Generalization in Robotic Manipulation Through The Use of Non-Rigid Registration. *International Symposium on Robotics Research*. 61

[Schulman et al., 2015] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *Intl. Conference on Machine Learning*. 21, 23, 103

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. 23, 103

[Schwab et al., 2019] Schwab, D., Springenberg, T., Martins, M. F., Lampe, T., Neunert, M., Abdolmaleki, A., Hertweck, T., Hafner, R., Nori, F., and Riedmiller, M. (2019). Simultaneously learning vision and feature-based control policies for real-world ball-in-a-cup. *arXiv preprint arXiv:1902.04706*. 23

[Shotton et al., 2013] Shotton, J., Glocker, B., Zach, C., Izadi, S., Criminisi, A., and Fitzgibbon, A. (2013). Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937. 17

[Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*. 22, 49, 103

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354. 103

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. 17

[Singh et al., 2014] Singh, A., Sha, J., Narayan, K. S., Achim, T., and Abbeel, P. (2014). Bigbird: A large-scale 3d database of object instances. In *IEEE Intl. Conference on Robotics and Automation*. 20, 105

[Snell et al., 2017] Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*. 78, 80, 113

[Stadie et al., 2017] Stadie, B. C., Abbeel, P., and Sutskever, I. (2017). Third-person imitation learning. *International Conference on Learning Representations (ICLR)*. 49

[Starek et al., 2015] Starek, J. A., Gomez, J. V., Schmerling, E., Janson, L., Moreno, L., and Pavone, M. (2015). An asymptotically-optimal sampling-based algorithm for bi-directional motion planning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2072–2078. IEEE. 19

[Steger, 2001] Steger, C. (2001). Similarity measures for occlusion, clutter, and illumination invariant object recognition. In *Joint Pattern Recognition Symposium*, pages 148–154. Springer. 16

[Sucan et al., 2012] Sucan, I. A., Moll, M., and Kavraki, L. E. (2012). The open motion planning library. *Robotics & Automation Magazine*. 111

[Şucan et al., 2012] Şucan, I. A., Moll, M., and Kavraki, L. E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82. `https://ompl.kavrakilab.org`. 124

[Sun et al., 2013] Sun, K., Aragon-Camarasa, G., Cockshott, P., Rogers, S., Siebert, J. P., Sun, L., Aragon-Camarasa, G., Cockshott, P., Rogers, S., and Siebert, J. P. (2013). A Heuristic-Based Approach for Flattening Wrinkled Clothes. *Towards Autonomous Robotic Systems)*. 63

[Sung et al., 2017] Sung, J., Lenz, I., and Saxena, A. (2017). Deep multimodal embedding: Manipulating novel objects with point-clouds, language and trajectories. *IEEE Intl. Conference on Robotics and Automation*. 79

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge. 22, 40

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9. 17

[Tamei et al., 2011] Tamei, T., Matsubara, T., Rai, A., and Shibata, T. (2011). Reinforcement Learning of Clothing Assistance with a Dual-arm Robot. *International Conference on Humanoid Robots*. 61

[Tang et al., 2012] Tang, J., Miller, S., Singh, A., and Abbeel, P. (2012). A textured object recognition pipeline for color and depth image data. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3467–3474. IEEE. 16

[Tassa et al., 2018] Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*. 101, 103, 104, 113, 117

[Tejani et al., 2014] Tejani, A., Tang, D., Kouskouridas, R., and Kim, T.-K. (2014). Latent-class hough forests for 3d object detection and pose estimation. In *European Conference on Computer Vision*, pages 462–477. Springer. 20

[Thananjeyan et al., 2017] Thananjeyan, B., Garg, A., Krishnan, S., Chen, C., Miller, L., and Goldberg, K. (2017). Multilateral surgical pattern cutting in 2D orthotropic gauze with deep reinforcement learning policies for tensioning. *International Conference on Robotics and Automation)*. 61

[Thrun and Pratt, 2012] Thrun, S. and Pratt, L. (2012). *Learning to learn*. Springer Science & Business Media. 77

[Tobin et al., 2017a] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017a). Domain randomization for transferring deep neural networks from simulation to the real world. *IROS*. 31, 48, 50, 86, 93

[Tobin et al., 2017b] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. (2017b). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. *International Conference on Intelligent Robots and Systems*. 62, 64

[Todorov et al., 2012a] Todorov, E., Erez, T., and Tassa, Y. (2012a). Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. 31

[Todorov et al., 2012b] Todorov, E., Erez, T., and Tassa, Y. (2012b). Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*. IEEE. 110

[Triantafillou et al., 2017] Triantafillou, E., Zemel, R., and Urtasun, R. (2017). Few-shot learning through an information retrieval lens. *Advances in Neural Information Processing Systems*. 78

[Tzeng et al., 2016] Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S., Saenko, K., and Darrell, T. (2016). Adapting deep visuomotor representations with weak pairwise constraints. *Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 49

[Uijlings et al., 2013] Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective search for object recognition. *International journal of computer vision*, 104(2):154–171. 17

[Ulbrich et al., 2011] Ulbrich, S., Kappler, D., Asfour, T., Vahrenkamp, N., Bierbaum, A., Przybylski, M., and Dillmann, R. (2011). The opengrasp benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation. In *IEEE Intl. Conference on Intelligent Robots and Systems*. 21, 105

[Van Hasselt et al., 2016] Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100. 23

[Varley et al., 2015] Varley, J., Weisz, J., Weiss, J., and Allen, P. (2015). Generating multi-fingered robotic grasps via deep learning. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 4415–4420. IEEE. 19

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008. 119, 122

[Vecerik et al., 2017] Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*. 23, 119

[Večerík et al., 2017] Večerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., and Riedmiller, M. (2017). Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *CoRR*. 61, 64, 65

[Vinyals et al., 2016] Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. (2016). Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*. 78, 113

[Wada et al., 2020] Wada, K., Sucar, E., James, S., Lenton, D., and Davison, A. J. (2020). Morefusion: Multi-object reasoning for 6d pose estimation from volumetric fusion. *IEEE Conference on Computer Vision and Pattern Recognition*. 18, 119, 123

[Wang et al., 2019] Wang, C., Xu, D., Zhu, Y., Martín-Martín, R., Lu, C., Fei-Fei, L., and Savarese, S. (2019). Densefusion: 6d object pose estimation by iterative dense fusion. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3343–3352. 18, 119, 123

[Wang et al., 2015] Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*. 23

[Watkins, 1989] Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge England. 43

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256. 119, 123

[Wilson and Martinez, 2003] Wilson, D. R. and Martinez, T. R. (2003). The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451. 36

[Wisspeintner et al., 2009] Wisspeintner, T., Van Der Zant, T., Iocchi, L., and Schiffer, S. (2009). Robocup@ home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*. 104

[Xiang et al., 2018] Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2018). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems*. 105

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Intl. Conference on Machine Learning*, pages 2048–2057. 119, 123, 128

[Yahya et al., 2016] Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., and Levine, S. (2016). Collective robot reinforcement learning with distributed asynchronous guided policy search. *arXiv preprint arXiv:1610.00673*. 22

[Yamakawa et al., 2011] Yamakawa, Y., Namiki, A., and Ishikawa, M. (2011). Motion planning for dynamic folding of a cloth with two high-speed robot hands and two high-speed sliders. *International Conference on Robotics and Automation*. 62, 63

[Yan et al., 2017] Yan, M., Frosio, I., Tyree, S., and Kautz, J. (2017). Sim-to-real transfer of accurate grasping with eye-in-hand observations and continuous control. *arXiv preprint arXiv:1712.03303*. 131

[Yang et al., 2015] Yang, Y., Li, Y., Fermuller, C., and Aloimonos, Y. (2015). Robot learning manipulation action plans by" watching" unconstrained videos from the world wide web. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 24, 90

[Yoon et al., 2004] Yoon, W.-K., Goshozono, T., Kawabe, H., Kinami, M., Tsumaki, Y., Uchiyama, M., Oda, M., and Doi, T. (2004). Model-based space robot teleoperation of ets-vii manipulator. *IEEE Transactions on Robotics and Automation*, 20(3):602–612. 15

[Yu et al., 2018] Yu, T., Finn, C., Xie, A., Dasari, S., Zhang, T., Abbeel, P., and Levine, S. (2018). One-shot imitation from observing humans via domain-adaptive meta-learning. *Robotics: Science and Systems*. 24, 90, 96, 101, 111

[Yu et al., 2019] Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Levine, S., and Finn, C. (2019). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. `https://meta-world.github.io`. 104

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer. 17

[Zeng et al., 2020] Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2020). Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*. 118

[Zeng et al., 2018a] Zeng, A., Song, S., Welker, S., Lee, J., Rodriguez, A., and Funkhouser, T. (2018a). Learning synergies between pushing and grasping with self-supervised deep reinforcement learning. In *IEEE Intl. Conference on Intelligent Robots and Systems*, pages 4238–4245. IEEE. 118

[Zeng et al., 2018b] Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., et al. (2018b). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE. 30

[Zhang et al., 2017] Zhang, F., Leitner, J., Milford, M., and Corke, P. (2017). Sim-to-real transfer of visuo-motor policies for reaching in clutter: Domain randomization and adaptation with modular networks. *world*, 7(8). 131

[Zhang et al., 2015] Zhang, F., Leitner, J., Milford, M., Upcroft, B., and Corke, P. (2015). Towards vision-based deep reinforcement learning for robotic motion control. *Australasian Conference on Robotics and Automation (ACRA)*. 47, 56

[Zhang et al., 2019] Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. (2019). Self-attention generative adversarial networks. In *Intl. Conference on Machine Learning*, pages 7354–7363. 119, 123

[Zhang et al., 2016a] Zhang, M., McCarthy, Z., Finn, C., Levine, S., and Abbeel, P. (2016a). Learning deep neural network policies with continuous memory states. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 520–527. IEEE. 22

[Zhang et al., 2016b] Zhang, T., Kahn, G., Levine, S., and Abbeel, P. (2016b). Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 528–535. IEEE. 21

[Zhang et al., 2018] Zhang, T., McCarthy, Z., Jow, O., Lee, D., Goldberg, K., and Abbeel, P. (2018). Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *International Conference on Robotics and Automation*. 23, 76

[Zhu et al., 2014] Zhu, M., Derpanis, K. G., Yang, Y., Brahmbhatt, S., Zhang, M., Phillips, C., Lecce, M., and Daniilidis, K. (2014). Single image 3d object detection and pose estimation for grasping. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3936–3943. IEEE. 16

[Zhu et al., 2018] Zhu, Y., Wang, Z., Merel, J., Rusu, A., Erez, T., Cabi, S., Tunyasuvunakool, S., Kramár, J., Hadsell, R., de Freitas, N., et al. (2018). Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*. 131