Visual Imitation Learning for Robot Manipulation

Maximilian Karl Rudolf Sieb

CMU-RI-TR-19-07

May 2019

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

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

Keywords: Visual Imitation Learning, Representation Learning, Reinforcement Learning, Robot Learning

For my family and friends who supported me along this journey.

Abstract

Imitation learning has been successfully applied to solve a variety of tasks in complex domains where an explicit reward function is not available. However, most imitation learning methods require access to the robot's actions during demonstration. This stands in a stark contrast to how we humans imitate: we acquire new skills by simply observing other humans perform a task, mostly relying on the visual information of the scene.

In this thesis, we examine how we can endow a robotic agent with this capability, i.e., how to acquire a new skill via *visual imitation learning*. A key challenge in learning from raw visual inputs is to extract meaningful information from the input scene, and enabling the robotic agent to learn the demonstrated skill based on the accessible input data. We present a framework that encodes the visual input of a scene into a factorized graph representation, casting one-shot visual imitation of manipulation skills as a visual correspondence learning problem. We show how we detect corresponding visual entities of various granularities in both the demonstration video and the visual input of the learner during the imitation process. We build upon multi-view self-supervised visual feature learning, data augmentation by synthesis, and category-agnostic object detection to learn scene-specific visual entity detectors. We then measure perceptual similarity between demonstration and imitation based on matching of the spatial arrangements of corresponding detected visual entities, encoded as a dynamic graph during demonstration and imitation.

Using different visual entities such as human keypoints and object-centric pixel features, we show how these relational inductive biases regarding object fixations and arrangements can provide accurate perceptual rewards for visual imitation. We show how the proposed image graph encoding drives successful imitation of a variety of manipulation skills within minutes, using a single demonstration and without any environment instrumentation.

Acknowledgments

I would like to thank my advisors Professor Fragkiadaki and Professor Kroemer for their extraordinary support during my two great years here at Carnegie Mellon University. They helped me shape my research passion and skills like no other individual, and they supported me throughout my journey through fruitful advice and exciting discussions. If it were not for them, I would not be in the position where I am right now. I sincerely thank them for their unconditional support and belief in my potential.

I am thankful for Professor Salakhutdinov's counsel along the way, and his advice on how to frame research questions in the unexplored field of deep learning in a structured manner. His classes also were my first exposure ever to deep learning, and his instructive way of teaching lay the foundations for my understanding of deep learning. I am glad to have had the opportunity to attend his lectures, and for all the fruitful discussions along the way. I also want to thank Lerrel Pinto for the great discussions about how to bring robotics out of an academic environment into the real world. I am also thankful to Professor Atkeson for providing great insights regarding robot learning in general, but specifically on model-based and compositional learning to bring our vision of bringing robotics into everyday life a little closer to realization.

Thanks to everyone in the IAM lab who has been with me along this journey. Specifically, I want to mention Samuel Clarke and Austin Wang, as we three were the first members of the lab founded by Professor Oliver Kroemer, and I am happy to not just call them colleagues, but also friends. I was fortunate enough to have been part of newly emerging lab, and it was a great experience to see a lab grow to the size it is today. I am more than looking forward to seeing great research and ideas coming from there in the future.

I want to express great thanks also to Xian Zhou who has become a close colleague and friend of mine during the last two semesters, and who has been a great to work with.

I am grateful for the funding received through the German Academic Scholarship Foundation, the German Academic Exchange Service, and the Ulderup Foundation. This Master's study would not have been made possible for me if it were not for their generous support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Humans acquire new skills with an ease that has yet to be matched by robots and other intelligent agents. One line of thought takes inspiration from humans' ability to bootstrap their learning process through observing other humans executing skills, and then trying to imitate the observed behavior. This imitation learning approach allows robots to acquire manipulation skills from human demonstrations. This way, a robot can be trained quickly in an intuitive manner by non-roboticists to perform different tasks. The demonstrations can be provided to the robot using a number of techniques, including teleoperation and kinesthetic teaching.

Imitation learning, also known as *Learning from Demonstration*, has been successfully applied to solve many different tasks in complex domains such as helicopter flight [1], playing table tennis [29], and accomplishing human-like reaching motions [9] among many others. While imitation learning has the benefit of comparatively fast skill acquisition, it requires high-quality demonstration of a human expert. In the area of robot manipulation, demonstrations are usually provided through kinesthetic teaching, in which a human expert physically guides the robot to perform a task by moving the robot arm around the work space. Compared to other techniques, imitation learning has the benefit of endowing a robotic agent with the capabilities of comparatively fast skill acquisition.

1.1 Motivation

Learning a skill from demonstration often requires access to the exact movements of the robot during the demonstration, i.e. the human must teleoperate the robot or kinesthetically teach the robot. It is still an open-ended research question of how we can teach skills to robotic agents without access to ground-truth actions and states. For example, in a home-setting where a robot is to prepare a meal from scratch, the human could provide a demonstration of the target skill. However, in the most natural setup, the location of the ingredients on the kitchen top are unknown, and the exact handling of the tools and ingredients by the human is also not accessible. A key challenge here is to extract meaningful information from the input scene, and enabling the robotic agent to learn the demonstrated skill based on the accessible input data.

The most intuitive method for a human to provide a demonstration is through *visual demonstrations*, i.e., the human performs the task as they would naturally, and the robot observes the human using a camera. Although this method is easy for the human, it does not allow the robot to observe the actions and forces employed by the human to perform the task. The embodiment of the human and robot is different, which means that directly mapping the human's motions and dynamics for performing the skill to the robot's body is not possible.

The goal of manipulation tasks is to change the states of objects in the environment in a predictable manner. The robot therefore does not need to mimic the movements of the human exactly. Instead, it should imitate the state trajectories of the objects in the scene, using the human demonstrator only as an additional supervisory signal during the learning process.

1.2 Contribution

We formulate the visual imitation problem as a reinforcement learning problem with a metric-based cost function, wherein the robot receives less cost if its actions result in object states that are more *similar* to those observed during the human demonstrations. Using this cost function, the robot can employ reinforcement learning to learn policies for imitating the demonstrated manipulations in different situations. This approach allows the robot to learn suitable manipulation policies despite the differing embodiment of the human and robot, i.e. we do not have access to ground truth actions of the human expert.

The key problem with employing this approach is defining what it means for two manipulations to be more or less *similar*. If we directly compared the full pixel observations of the scenes, then even two distinct manipulations might be considered as similar because they were both performed in front of the same background. Instead, the robot should focus on the subset of relevant objects in the scene and ignore the irrelevant ones [30]. The similarity measure should also compare how the states of the relevant objects change over the course of the manipulation. These changes in state may correspond to local changes in the individual objects' appearances as well as their are arrangement in the scene. In this thesis, we cast the problem as minimizing a simple metric-based distance in a feature space that is composed of different visual entities.

In this work, we attempt to close the gap between deep object detectors used in computer vision and machine perception based approaches in robotic learning. Since object instances encountered by a robotic agent are often task-specific and not part of the categories labelled in commonly available computer vision datasets, the successful application of deep object detectors in this settings is often challenging. Instead, frame-wide representations are preferred, both for reinforcement learning [4, 41] and for visual imitation [57]. We show that generating a synthetic dataset based on a few ground truth examples, commonly referred to as data dreaming, is sufficient to train on-the-fly object detectors. Even though they do not detect each target object in *all* possible contexts, they are powerful enough in the current scene to track the object through partial occlusions.

Furthermore, we propose object-centric & dynamic hierarchical graph representations for demonstration and imitation videos, where nodes represent visual entities tracked in space and time, and edges represent their 3D spatial arrangements. Our graphical image encoding is based on the simple observation that the *appearance* of the scene, in some level of granularity (objects, parts or pixels), remains constant throughout a demonstration, but the *spatial relationships* of the objects, parts or pixels change over time–a concept which follows directly from laws of Newtonian physics. Our graph representations capture such what-where decomposition: the node visual entities, which are in one-to-one correspondence between the teacher and learner environments (Figure 1.1) persist in time, but their spatial arrangements may change. We dynamically consider edges between node entities that are in motion in the demonstration video, and in this way focus the cost function to attend to the relevant part of the scene. Our imitation cost function is based on matching of the spatial distances between corresponding nodes, and it guides reinforcement learning of manipulation tasks from a single video demonstration and using only a handful of real-world interactions.

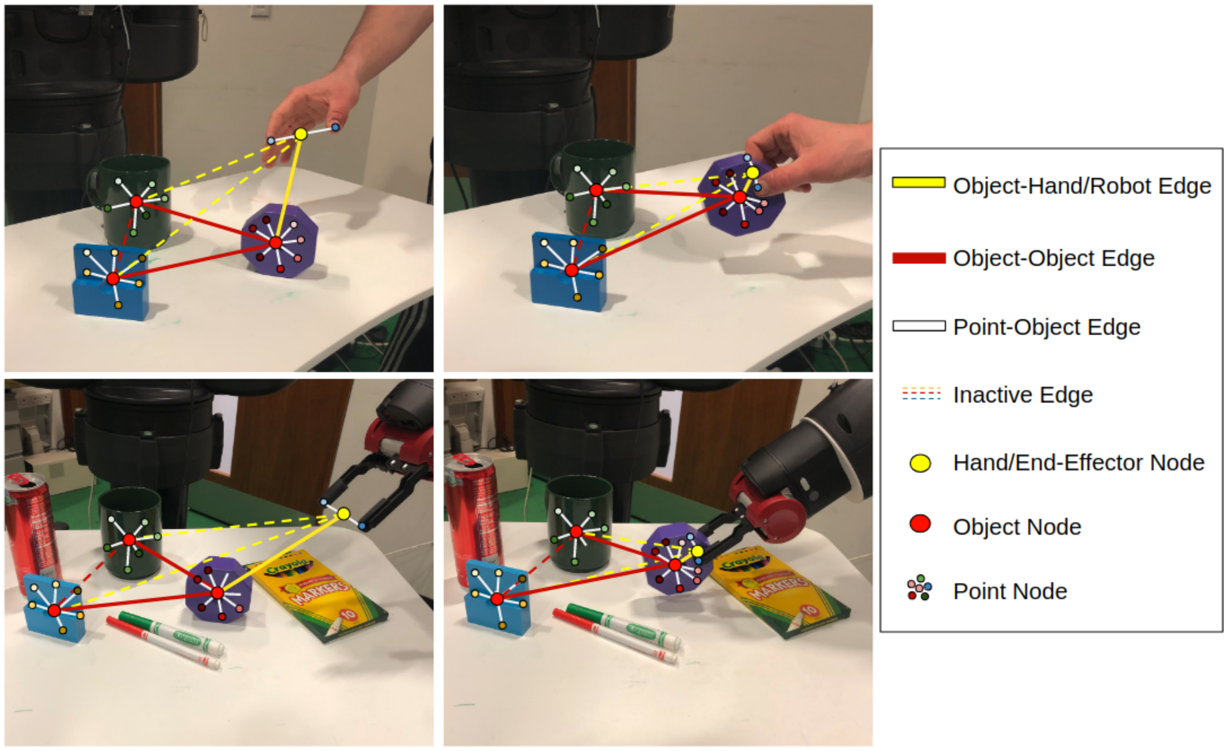

Figure 1.1: **Learning by watching humans using visual entity graphs (VEG).** We show the VEGs of a human demonstration and robot imitation for 2 timesteps of a typical imitation. Corresponding nodes in the human demonstration (top) and robot imitation (bottom) share the same color. Our graphs are hierarchical, and there exist object-robot/hand edges, object-object edges, and object-point feature nodes. Active edges are depicted in bold, inactive edges in dashed lines. Edges are dynamically activated over time based on motion attention, and the cost function of our imitation method is based on matching the spatial arrangements along activated corresponding edges in the graph. Our graph representation is robust against viewpoint variation and cluttered backgrounds.

The learning problem then is precisely the discovery of corresponding visual entities between the teacher's and learner's environments. This requires fine-grained visual understanding of the demonstrator's and robot's environments. The objects used by the demonstrator are often not included in the labelled object categories of ImageNet [12] or MS-COCO [37], and thus, pre-trained object-detectors are not ad-hoc not particularly useful here. In this work, we build upon human keypoint detectors, self-supervised cross-view visual feature learning, and synthetic data dreaming for training detectors on-the-fly to obtain *scene-conditioned visual entity detectors* which establish correspondences between the teacher's and learner's workspace, despite the difference in embodiment between the robot and the human, different occlusion patterns, and robot-human body visual discrepancies (Figure 1.1). The proposed visual entity detectors, though they may not localize the target entities in *all* possible contexts, they can effectively track the objects through partial occlusions in the *current* scene context of the learner and teacher.

The proposed framework was evaluated both in a simulator on several pick-and-place tasks, as well as on a Baxter robot putting rings onto a pole and rotating an object. Our experiments suggest object-centric state representations outperform frame-wide ones [57], which do not introduce any structural biases regarding object-centric attention. Agreeing with authors of [8], incorporating useful structural biases can greatly accelerate learning of neural architectures and alleviate the need of obtaining a large enough dataset to train these architectures. Structural bias via placing hard attention on relevant objects greatly aids the interpretability of the algorithm compared to frame-centric end-to-end approaches.

1.3 Outline

We will first discuss about required background material related to the presented approach, including trajectory optimization and neural architectures in the domain of computer vision.

Secondly, we will talk about existing approaches in the area of visual imitation learning and object-centric reinforcement learning which most closely relate to our proposed approach.

We will then outline our method of creating robust object-centric visual representations, *Visual Entity Graphs for Visual Imitation Learning*, and we will show how we can utilize this graph-like representational framework in a reinforcement learning setup.

Finally, we will demonstrate the validity of our approach by showing a variety of different manipulation tasks that were learned using our proposed framework.

Chapter 2

Background

2.1 Object Detection and Instance Segmentation

$\mathcal{G}_t$

Deep learning methods have been very successful in solving data-driven tasks in recent years, most prominently in the area of computer vision. A typical application scenario of deep learning in this domain is object detection: The goal is to find all objects within an image, given a finite set of known objects. In most cases, these objects are localized via 2D bounding boxes, without providing detection on a pixel level. Instance segmentation aims to tackle exactly this problem by also predicting the pixel-level segmentation for a given object, i.e., the goal is to predict bounding boxes and corresponding binary segmentation masks for each detected instance of every object that is contained in the annotation set, as illustrated in Figure 2.2. State-of-the-art instance segmentation networks such as Mask R-CNN [21] build upon object detection architectures such as Faster R-CNN [50] by adding a fully connected layer to additional predict a binary segmentation mask for every predicted bounding box, as shown in Figure 2.2.

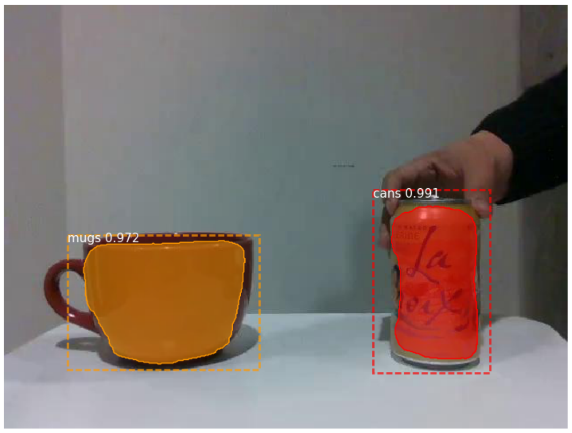

Figure 2.1: Instance segmentation involving a human performing a pouring task.

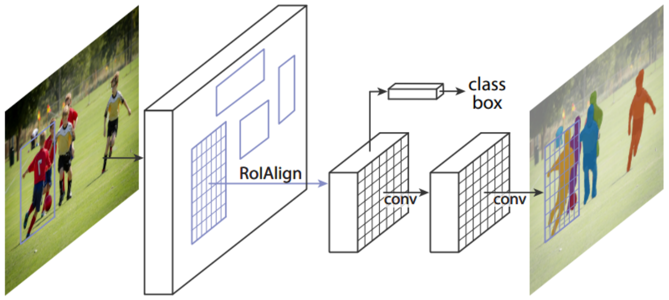

Figure 2.2: Mask R-CNN pipeline. [21]

2.2 Reinforcement Learning

In reinforcement learning [60], an agent interacts with an environment by executing actions and collecting rewards. In most cases, the agent's goal is maximizing the accumulated reward over its lifetime by executing appropriate actions. In a typical reinforcement learning setup, we assume the environment to be static, i.e., the environment dynamics do not change over time, although they can be of stochastic nature.

This setting can be formally described via a Markov Decision Process (MDP), where an MDP is defined as $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho_0, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ the state transition probability distribution, $\rho_0 : \mathcal{S} \mapsto [0, 1]$ the initial state distribution, and $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ the defined reward function. Typically,the goal is to find a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$ that maximizes the expected return $R = \mathbb{E}_{\boldsymbol{x}_0 \sim \rho_0, \boldsymbol{u}_t \sim \pi(\cdot|\boldsymbol{x}_t), \boldsymbol{x}_{t+1} \sim \mathcal{P}(\cdot|\boldsymbol{x}_t, \boldsymbol{u}_t)} \left[\sum_{t=0}^{T-1} \gamma^t r\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) \right]$, where $\boldsymbol{x}_0 \sim \rho_0, \boldsymbol{u}_t \sim \pi\left(\cdot|\boldsymbol{x}_t\right)$, and $\boldsymbol{x}_{t+1} \sim \mathcal{P}\left(\cdot|\boldsymbol{x}_t, \boldsymbol{u}_t\right)$. T can be both finite or infinite, and the discount factor $\gamma \in [0, 1)$ ensures that the return stays bounded for an infinite time-horizon T. In a control-theoretic setting, using cost instead of reward is common in case where cost is simply defined as the negated reward.

2.3 Policy Search using Time-Varying Linear-Gaussian Controllers

Policy search tackles the problem of finding the optimal policy for the reinforcement learning problem defined above. More formally, we are trying to optimize the total cost

$$J\left(\mathbf{x}_0, \mathbf{U}\right) = E_{\pi_\theta}\left[\sum_{t=1}^{T} \ell\left(\mathbf{x}_t, \mathbf{u}_t\right)\right] \tag{2.1}$$

of a policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, which is a distribution over actions $\mathbf{u}_t$, conditioned on the state $\mathbf{x}_t$ for a given time step t, and $\mathbf{U} := \{\mathbf{u}_0, \mathbf{u}_1 \ldots, \mathbf{u}_{N-1}\}$. The term $\ell(\mathbf{u}_t, \mathbf{x}_t)$ represents the cost for the current time step t. The *optimal cost-to-go* for the current time step t is denoted by $V_t(\mathbf{x}) = \min_{\mathbf{U}_t} J_t\left(\mathbf{x}, \mathbf{U}_t\right)$, where $\mathbf{U}_t := \{\mathbf{u}_t, \mathbf{u}_1 \ldots, \mathbf{u}_{N-1}\}$.

Instead of directly optimizing π_θ, we can define a low-level control policy $\pi(\mathbf{u}_t|\mathbf{x}_t, \theta)$ and an upper-level search distribution $\pi(\theta|\omega)$. For example, we could parametrize our low-level control policy as a simple Gaussian state-feedback controller via $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) = \mathcal{N}(f_\omega(\mathbf{x}_t), \sigma^2 I)$, where $\theta = (\omega, \sigma)$.

If we parametrize $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$ as a time-varying linear-Gaussian policy, we can take advantage of efficient optimization techniques that allow for achieving locally optimal policies by optimizing a specific state trajectory. The parametric form of π_θ in this case would be $\pi(mbu_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{C}_t)$, where $\mathbf{k}_t$ is a time-dependent feedback gain matrix, k_t a time-dependent feedback offset, and $\mathbf{C}_t$ a time-dependent noise-term. More specifically, we then aim to learn a set of policy parameters that optimizes an initial trajectory by repeatedly computing an optimal set of linear-Gaussian control parameters. This process is often referred to as *trajectory optimization* [22], which aims to find a policy that locally minimizes a given cost function. One specific method i

2.3.1 Model-Based Trajectory Optimization

For general nonlinear dynamical systems, we can utilize Differential Dynamic Programming (DDP) [40] to perform trajectory optimization by computing the Linear Quadratic Regulator solution to a linear approximation of the dynamics and second-order approximation of the cost at the current time step. We then iteratively refine this solution taking into account the cost-to-go of the remaining trajectory. This approach is also known as iterative-Linear-Quadratic Regulator (iLQR) [35]. Formally, we compute approximations for the, potentially *a-priori* unknown, nonlinear transition dynamics as well as for the value function V and the action-value function Q with

$$V(\mathbf{x}_t) = \frac{1}{2}\mathbf{x}_t^{\mathrm{T}} V_{\mathbf{x},\mathbf{x},t}\mathbf{x}_t + \mathbf{x}_t^{\mathrm{T}} V_{\mathbf{x},t} + \mathrm{const} \tag{2.2}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2}[\mathbf{x}_t; \mathbf{u}_t]^{\mathrm{T}} Q_{\mathbf{xu},\mathbf{xu},t}[\mathbf{x}_t; \mathbf{u}_t] + [\mathbf{x}_t; \mathbf{u}_t]^{\mathrm{T}} Q_{\mathrm{xu},t} + \mathrm{const} \tag{2.3}$$

The first and second-order derivative terms can be recursively computed via

$$Q_{\mathbf{xu},\mathbf{xu},t} = \ell_{\mathbf{xu},\mathbf{xu},t} + f_{\mathbf{xu},t}^{\mathrm{T}} V_{\mathbf{x},\mathbf{x},t+1} f_{\mathbf{xu},t} \tag{2.4}$$

$$Q_{\mathbf{xu},t} = \ell_{\mathbf{xu},t} + f_{\mathbf{xu},t}^{\mathrm{T}} V_{\mathbf{x},t+1} \tag{2.5}$$

$$V_{\mathbf{x},\mathbf{x},t} = Q_{\mathbf{x},\mathbf{x},t} - Q_{\mathbf{u},\mathbf{x},t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u},t}^{-1} Q_{\mathbf{u},\mathbf{x}} \tag{2.6}$$

$$V_{\mathbf{x},t} = Q_{\mathbf{x},t} - Q_{\mathbf{u},\mathbf{x},t}^{\mathrm{T}} Q_{\mathbf{u},\mathbf{u},t}^{-1} Q_{\mathbf{u},t} \tag{2.7}$$

It can now be shown that the linear Gaussian policy $\pi(\mathbf{u}_t|\mathbf{x}_t) = \hat{\mathbf{u}}_t + \mathbf{k}_t + \mathbf{K}_t(\mathbf{x}_t - \hat{\mathbf{x}}_t)$ minimizes this approximated Q function with $\mathbf{K}_t = -Q_{\mathbf{u},\mathbf{u},t}^{-1} Q_{\mathbf{u},\mathbf{x},t}$ and $\mathbf{k}_t = -Q_{\mathbf{u},\mathbf{u}t}^{-1} Q_{\mathbf{u},t}$. Here, $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ are the action and state trajectory of the last policy rollout.

In the case of unknown (nonlinear) dynamics, the conventional iLQR approach as discussed above refines the optimal action sequence $\mathbf{U}$ via line-search with

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 \tag{2.8}$$

$$\hat{\mathbf{u}}_t = \mathbf{u}_t + \alpha \mathbf{k}_t + \mathbf{K}_t \left(\hat{\mathbf{x}}_t - \mathbf{x}_t\right) \tag{2.9}$$

$$\hat{\mathbf{x}}_{t+1} = \mathbf{f}\left(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t\right) \tag{2.10}$$

The newly computed action sequence $\hat{\mathbf{U}}$ is then used to iteratively improve the Q approximation above until convergence.

2.3.2 Trajectory Optimization with Unknown Transition Dynamics

If the dynamics are unknown, which is the case in many real-world robotics applications, we can approximate the dynamics locally assuming a linear-Gaussian dynamics model:

$$p\left(\mathbf{x}_{t+1} | \mathbf{x}_t, \mathbf{u}_t\right) = \mathcal{N}\left(f_{\mathbf{x},t}\mathbf{x}_t + f_{\mathbf{u},t}\mathbf{u}_t + f_{c,t}, \mathbf{\Sigma}_t\right) \tag{2.11}$$

where $\mathbf{\Sigma}$ represents the noise of the approximated dynamical system, and $f_{c,t}$ represents the bias term [19].

We can then fit these dynamics by rolling out the current policy N times for T time steps. This yields n trajectory tuples $\{\tau\}_{i=1,\ldots,N}$ with $\tau_i = \{\mathbf{x}_{1i}, \mathbf{u}_{1i}, \ldots, \mathbf{x}_{Ti}, \mathbf{u}_{Ti}\}$, which we use to fit the linear-regression problem in 2.11.

Since we rely on actual system rollouts to evaluate our approximated control sequence, we cannot employ the line-search procedure laid out in 2.8ff. This would require prohibitively large amount of rollouts of the actual policy for every iteration. However, this procedure is necessary to restrict the updated policy to stay close to the previous policy. Another way of restricting the new policy to be close to the old one is to incorporate an additional constraint on the new policy via bounding the Kullback-Leibler divergence between the old and new policy [33]. More formally, the new constrained optimization problem can be written as

$$\min_{p(\tau) \in \mathcal{N}(\tau)} E_p[J(\tau)] \text{ s.t. } D_{\mathrm{KL}}(p(\tau) \| \hat{p}(\tau)) \leq \epsilon \tag{2.12}$$

The Lagrangian of this constrained optimization problem is given by

$$\mathcal{L}(p(\tau), \eta) = E_p[J(\tau)] + \eta \left[D_{\mathrm{KL}}(p(\tau) \| \hat{p}(\tau)) - \epsilon\right] \tag{2.13}$$

and can for example be solved via dual gradient descent,

We can further augment this deterministic policy by adding noise, using the curvature of the current Q-function estimate as an indicator for how explorative the policy should be at the current time step. We then obtain the following linear-Gaussian controller as our parametrized policy

$$\pi\left(\mathbf{u}_t | \mathbf{x}_t\right) = \mathcal{N}\left(\hat{\mathbf{u}}_t + \mathbf{k}_t + \mathbf{K}_t\left(\mathbf{x}_t - \hat{\mathbf{x}}_t\right), \mathbf{\Sigma}_t\right) \tag{2.14}$$

with $\mathbf{K}_t = -Q^{-1}_{\mathbf{u},\mathbf{u},t} Q_{\mathbf{u},\mathbf{x},t}$, $\mathbf{k}_t = -Q^{-1}_{\mathbf{u},\mathbf{u}t} Q_{\mathbf{u},t}$, $\mathbf{\Sigma}_t = Q^{-1}_{\mathbf{u},\mathbf{u},t}$, while $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{u}}_t$ represent the action and state trajectory of the last policy rollout.

Additionally, we can take into account a dynamics prior, for example via a Gaussian Mixture Model, to continuously update a global dynamics model of the environment. This way, we bootstrap the learning of the dynamical system transition parameters from previous rollouts.

For more details, refer to [33].

2.3.3 Combining Model-Free and Model-Based Trajectory Optimization

Model-based trajectory optimization methods such as iLQR are prone to getting stuck in local optima when presented an environment with discrete jumps in the local dynamical transition model, for example by switching between free-space and contact-constrained motion. Model-free methods do not suffer from this problem since they compute their policy parameters from the incurred cost only, abstracting away the environment dynamics. One prominent way of performing model-free trajectory optimization is via path integrals. More formally, we generate N rollouts with our current policy $\pi_\theta(\mathbf{u}_t|\mathbf{x}_t)$, and compute the probability weights

$$P\left(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}\right) = \frac{\exp\left(-\frac{1}{\eta_t} S\left(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}\right)\right)}{\int \exp\left(-\frac{1}{\eta_t} S\left(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}\right)\right) d\mathbf{u}_{i,t}} \tag{2.15}$$

for every time step and trajectory. Here, $S\left(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}\right) = \sum_{j=t}^{T} \ell\left(\mathbf{x}_{i,j}, \mathbf{u}_{i,j}\right)$ is the *sampled cost-to-go* of trajectory i for time step t.

We update our policy parameters by employing Maximum Likelihood Estimation after reweighing each previous control $\mathbf{u}_{i,t}$ with its corresponding probability weight $P\left(\mathbf{x}_{i,t}, \mathbf{u}_{i,t}\right)$. This essentially updates the parameters in a way so that control outputs that led to low cost will be more likely than control outputs that led to high cost. We can choose an arbitrary parametetrization of the control policy π_θ. By parametrizing it as a time-varying linear Gaussian controller as in the previous chapter, we can take advantage of combining both methods to employ a hybrid approach of model-based and model-free trajectory optimization methods. In this work, we will make use of such a hybrid approach, namely PILQR [10]. This is a state-of-the-art trajectory optimization method that combines, in a principled manner, the model-based generalization of linear quadratic regulators together with path integral policy search's [62] flexibility and ability to handle non-linear dynamics.

2.4 Imitation Learning

Imitation learning tackles the problem of learning a policy by imitating a not necessarily optimal expert demonstration defined by state-action tuples $\zeta = \{(s_0, a_0), (s_1, a_1), \ldots, (s_N, a_N)\}$. One prominent technique in this domain is behavioral cloning (BC) [6], which directly regresses over state-action tuples of given demonstration of an expert policy to obtain the imitation policy. Other approaches such as inverse reinforcement learning (IRL) try to first infer a reward function, and then obtain the optimal imitation policy by applying standard reinforcement learning using the inferred reward function. Both BC and IRL suffer from the correspondence problem, i.e. the expert and imitator operate in different action spaces, and the action sequence of the expert cannot be easily mapped to an action sequence of the imitator.

If the imitation learning process aims to only imitate the state-trajectory $\zeta = \{s_0, s_1, \ldots, s_N\}$ of an expert without direct access to the actions, we refer to this problem as *imitation from observation* [38]. In this setup, the goal is to learn a policy π_I that, given a set of demonstrated state-trajectories $D = \{\zeta_1, \ldots\zeta_N\}$, leads to behavior that produces to the same state-occupancy as the expert's state-trajectory distribution. In a one-shot learning setup as in this work, the problem boils down to learning an imitation policy π_I that follows the same state trajectory as the expert policy π_E for a sufficiently similar starting position in the given state space. Generally, imitation from observation does not assume that the expert and learner operate in the same context, i.e., the learner operates in a state space that differs from the expert's state space. However, in this work we assume that the learner and expert operate in the same state-space. When operating on visual input data, i.e., the setting of *visual imitation learning*, the learning problem also includes finding a useful state representation that can be used to compare the state-trajectories of the expert and the imitator, since operating on raw pixels would lead to a prohibitively large state space.

Chapter 3

Related Work

3.1 Visual Imitation Learning

Imitation learning concerns the problem of acquiring skills by observing demonstrations. It has been considered an integral part in the field of robotics, with the potential to highly expedite trial-and-error learning [55]. However, the vast majority of previous approaches assumes that such demonstrations are given in the work space of the agent, (e.g., through kinesthetic teaching or teleoperation in the case of robots [5, 25]) and the actions/decisions of the demonstrator can be imitated directly, either with behavioral cloning (BC) [67], inverse reinforcement learning (IRL) [45], or generative adversarial imitation learning (GAIL) [23].

The problem of mimicking humans based on visual information is challenging due to the difficulty of the visual inference needed for fine-grained activity understanding [59]. In this case, imitation requires a mapping between observations in the demonstrator space to observations in the imitator space [44]. Works like [38] collect multiple expert demonstrations and propose an imitation learning method based on video prediction and deep reinforcement learning, relying on full-frame image encodings as latent state space representations. [64] try to infer the expert's action sequence which is not available during the imitation process, while operating on a known state-space instead of visual input data. Approaches like [66] aim to overcome the correspondence problem by having a human expert kinesthetically teach a robot via virtual reality. Numerous works circumvent the difficult perception problem using special instrumentation of the environment to read off object and hand poses during video demonstrations, such as AR tags, and use rewards based on known 3D object goal configurations [49]. Other works use hand-designed reward detectors that only work on restrictive scenarios [42].

Recent approaches bypass explicit state estimation and attempt to imitate the demonstrator's behavior by directly matching latent perceptual states to which the input images are mapped. Numerous loss functions have been proposed to learn such an image or image-sequence encoding. One single RGB frame or an entire sequence of frames is encoded into an embedding vector using a combination of forward and inverse dynamics model learning in [2, 47], multiview invariant and time-contrastive metric learning in [57], or reconstruction and temporal prediction objectives in [16, 65]. [32] provides an overview of common loss metrics and inductive biases for state representation learning.

Having obtained this latent state space, imitation is carried out by iterative application of the inverse model in [2, 47], or via trajectory optimization in [57], where the state is the combination of the latent visual state embedding vector and the robot configuration. Our work is most similar to [57], in that we use a similar trajectory optimization method [10] for imitating a human visual demonstration and similar unsupervised loss functions for visual feature learning. However, we apply such metric learning to features collected within each relevant object bounding box, as opposed to a video frame, i.e., we share neural feature extraction weights across objects in the scene.

Utilizing graph encodings of a visual scene in terms of objects or parts and their pairwise relations has been found beneficial for generalization of action-conditioned scene dynamics [7, 18], and body motion and person trajectory forecasting [3, 26]. In contrast to the work of [16] which learns to represent an RGB image in terms of spatial coordinates of few keypoints, we employ explicit attention only to relevant objects, and preserve their correspondence in time, i.e., the detectors *bind* with specific objects in the scene.

Synthetic Data Augmentation: Synthetic data augmentation has been very successful in multiple machine perception tasks [20, 46, 53, 63]. In [20], images with cross-person occlusions are generated to help train occlusion-aware detectors. Context-specific data augmentation was used in [28] to train neural networks for object video segmentation. The work of [63] showed that domain randomization, namely augmenting textures of the background and the objects, even in a non-visually plausible way, helps in training successful detectors that work on real images.

3.2 Object-Centric Reinforcement Learning

Representing a visual observation in terms of objects or parts and their pairwise relations has been found beneficial for generalization of action-conditioned scene dynamics [7, 18, 27], body motion and person trajectory forecasting [3, 26], and reinforcement learning [14]. Such graph-encodings have also been used to learn a model of the agent [54], and use it for model-predictive control in non-visual domains. Devin et al. [13] use pre-trained object detectors and learn attention of the obtained detection boxes, which they incorporate as part of the state representation for policy learning. The graph representation we propose in this work not only employs explicit attention to relevant objects, parts, and points, but also preserves their correspondence in time, i.e., the detectors *bind* with specific objects, parts, points in the scene.

Chapter 4

Visual Entity Graphs for Visual Imitation Learning

We propose a graph-structured state representation for visual imitation learning. We leverage geometric relations between different types of *visual entities* in the scene, and we frame the problem of visual imitation learning as matching state trajectories using our proposed state representation. In this chapter, we will first formalize our proposed state representation, and define the different visual entities that we make use of. We will then lay out how we detect each proposed visual entity, and then show how we can use our state representation for learning policies from a single human demonstration video.

4.1 Formulation

We encode a demonstration video of length T provided by the human expert, and an imitation video of the same length T provided by the robotic agent in terms of two graph sequences $\mathcal{G}_D^t = \{\mathcal{V}_D^t, \mathcal{E}_D^t\}, t = 1 \cdots T$ and $\mathcal{G}_I^t = \{\mathcal{V}_{I,}^t, \mathcal{E}_I^t\}, t = 1 \cdots T$, where a node $\mathcal{V}_i^t$ corresponds to a visual entity and its respective 3D (X, Y, Z) world coordinate, and an edge $\mathcal{E}^{t,(i,j)}$ correspond to a binary attention between two nodes, indicating whether a relation between two node entities is present or not. We define a visual entity node $\mathcal{V}_i^t$ to be any object, object part, or point that can be reliably detected in the demonstrator's and imitator's work space. Note that the entity itself is time dependent, i.e. can dynamically appear and disappear over time. The only requirement is that each entity associated with the demonstration sequence must have a corresponding entity in the imitation sequence, i.e., $\forall \mathcal{V}_i^t \in \mathcal{V}_D^t \; \exists! \; \mathcal{V}_j^t \in \mathcal{V}_I^t \; s.t. \; \mathcal{V}_i^t \,\widehat{=}\, \mathcal{V}_j^t$, where correspondence "$\widehat{=}$" in this context is established via an *a priori* defined label-set. For example, when imitating a pouring task using the same *bottle* both during demonstration and imitation, both bottle instances would be in correspondence. If one uses a different instance of a bottle during imitation, which could mean different in terms of shape or visual appearance, then this new bottle instance would be in correspondence with the bottle instance of the demonstration.

We consider three types of node entities: object nodes $\mathcal{V}_o$, point nodes $\mathcal{V}_p$, and hand/robot nodes $\mathcal{V}_h$. Object nodes represent any rigid or non-rigid object that constitutes a separate physical entity in the world, while point nodes represent any visual or physical sub-part of any the associated visual entity, for example different pixels, as seen in 1.1. Hand/robot nodes represent the manipulation agent in the scene. All nodes are in one-to-one correspondence between the demonstrator and imitator graph. We do not consider edges between point nodes, as shown in Figure 4.1. Rather, each point node is connected only to the object node it is part of. In that sense, our graphs are hierarchical.

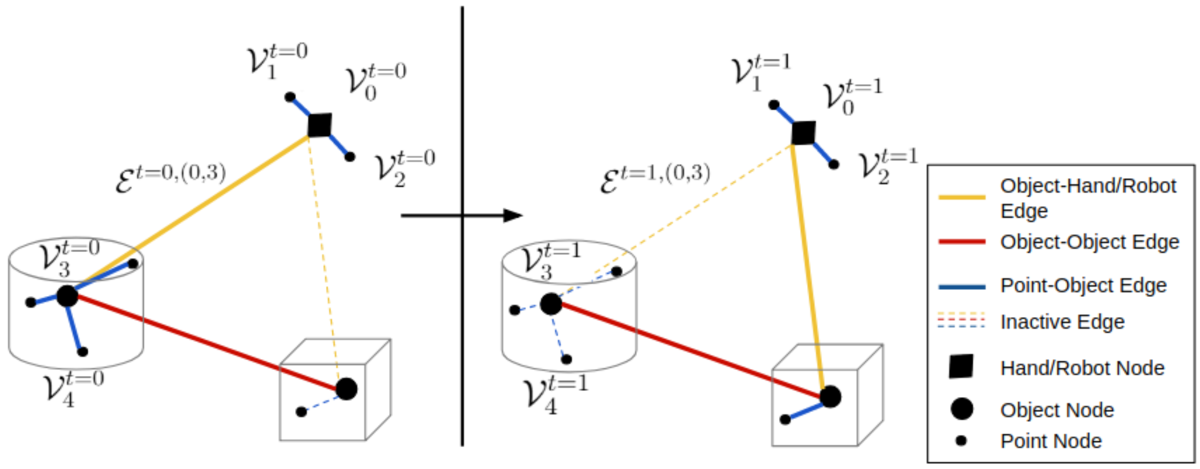

Figure 4.1: **Generic visual entity graph with object, hand and point nodes.** Depicted is a visual entity graph over two time steps. For illustration purposes, only five of overall nine entity nodes are depicted. Here, $\{\mathcal{V}_0^t\} \subseteq \mathcal{V}_h$, $\{\mathcal{V}_3^t\} \subseteq \mathcal{V}_o$, $\{\mathcal{V}_1^t, \mathcal{V}_2^t, \mathcal{V}_3^t\} \subseteq \mathcal{V}_p$, $\mathrm{att}(\mathcal{E}^{t=0,(0,3)}) = 1$, and $\mathrm{att}(\mathcal{E}^{t=1,(0,3)}) = 0$. Left: The cylinder is currently the anchor, and all edges connecting to it are active. Hand/Robot Edges are by default active. Right: The cube is now the anchor, and thus, all edges connecting to it become active, while the object-hand/robot edge of the cylinder becomes inactive as well as its point-object edges.

Visual entities can be tracked in time both in demonstration and imitation—although not necessarily across the entire video sequence—and, thus, certain nodes that are consecutive in time are also in correspondence. In principle, object nodes $\mathcal{V}_o$ and hand/robot nodes $\mathcal{V}_h$ together form a clique within the graph, while each object node $\mathcal{V}_o$ constitutes a *star* along with the point nodes $\mathcal{V}_p$ that belong to the respective object node. Edges in $\mathcal{G}_D$ and $\mathcal{G}_I$ are dynamically activated and deactivated over time, based on a motion attention function att, which encodes motion attention over the visual entities of the demonstrator. Motion attention in this context is defined as putting attention on objects that are in motion relative to other objects. In this context, we refer to these objects as .

Our cost function at each time step t measures visual dissimilarity across the demonstrator and imitator graphs $\mathcal{G}_D$ and $\mathcal{G}_I$ in terms of relative spatial arrangements of corresponding entities, as follows:

$$\mathcal{C}(\mathcal{G}_D^t, \mathcal{G}_I^t) = \sum_{i,j,i<j} w(\mathcal{E}^{t,(i,j)}) \cdot \mathrm{att}(\mathcal{E}^{t,(i,j)}) \cdot \left\| \left(\mathbf{x}_D^{t,i} - \mathbf{x}_D^{t,j}\right) - \left(\mathbf{x}_I^{t,i} - \mathbf{x}_I^{t,j}\right) \right\|, \tag{4.1}$$

where $\mathbf{x}_D^{t,i}$ denotes the 3D (X, Y, Z) world coordinate of the ith entity in the demonstrator's graph $\mathcal{G}_D$ at time t, $w(\mathcal{E}^{t,(i,j)}) \in \mathbb{R}$ denotes edge weights, and $\mathrm{att}(\mathcal{E}^{t,(i,j)}) \in \{0, 1\}$ is a binary, motion-guided attention function that determines whether a particular edge is present depending on the motion of the corresponding nodes. We consider the following scenarios to automatically determine whether an edge is active, i.e., $\mathrm{att}(\mathcal{E}^{t,(i,j)}) = 1$:

1. Edges are active if they connect to an anchor object. This true for both object-hand/robot edges as well as object-point edges.
2. Point-hand/robot edges are always active to correctly infer the gripper state of the robot if applicable. In our case, point nodes of the robot correspond to its fingertips.

Since we neither tie edges between point nodes $\mathcal{V}_p$ across different objects nor within objects themselves, only for point nodes that are connected to the object node they are part of visually.

Figure 4.1 illustrates this graphical inductive bias.

We tie weights across all edges of the same type, namely, object-hand edges, object-object edges and object-point edges. In that sense, the cost function encodes the relative distances in 3D between all entities that for which the binary attention function $\mathrm{att}(\mathcal{E}^{t,(i,j)})$ evaluated to 1 for the associated edge $\mathcal{E}^{t,(i,j)}$.

We assume no access to human annotations that would mark relevant corresponding entities across demonstrator's and imitator's environments. Learning boils down to training detectors for visual entities that can be reliably recognized between the demonstrator's and imitator's work space and also be reliably tracked over time.

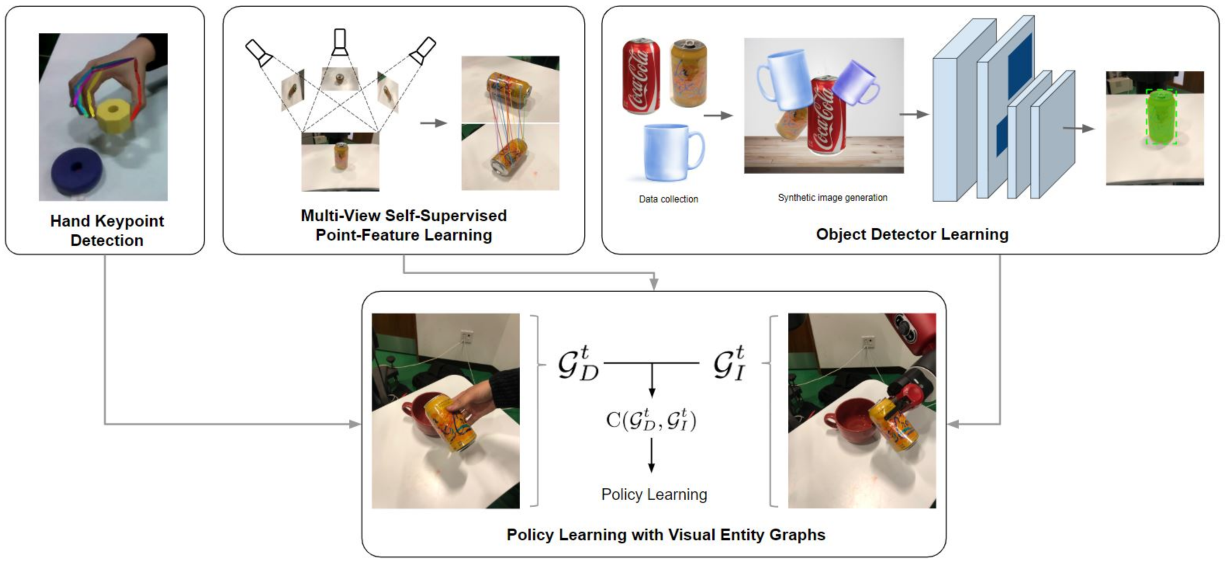
Hand Keypoint Detection
Multi-View Self-Supervised Point-Feature Learning
Data collection
Synthetic image generation
Object Detector Learning
$\mathcal{G}_D^t$
$\mathcal{G}_I^t$
$C(\mathcal{G}_D^t, \mathcal{G}_I^t)$
Policy Learning
Policy Learning with Visual Entity Graphs

Figure 4.2: **Detecting visual entities.** We use human hand keypoint detectors, multi-view pixel feature learning, and synthetic image generation for on-the-fly object detector training from only a few object mask examples. Using a manually designed mapping between the human hand and the robot, the visual entity detectors can effectively bridge the visual gap between demonstrator and imitator environment, are robust to background clutter, and generalize across different object instances.

4.2 Detecting Visual Entities

Since our definition of a visual entity included detectability of such an entity, we will now take a more detailed look at how such detection is implemented. In general, objects that the human demonstrator manipulates may not be part of the MS-COCO [36] object-labelled dataset. Moreover, for fine-grained object manipulation, such as object rotation, the motion of the object's bounding box would be uninformative. Instead, points on the surface of the object and their motion should be used. This is a design choice in our representation: instead of learning box appearance features that encode object rotation or deformation, we opt for pixel feature detectors which establish fine-grained correspondences to reveal the object's rotation or deformation without further training.

We detect object entities $\mathcal{V}_o$ via synthetic data augmentation for on-the-fly object detectors, and hand entities $\mathcal{V}_h$ during demonstration via human hand detectors. As for point entities $\mathcal{V}_p$, we opt for taking a set of pixels that we sample uniformly across each detected object entity, and we correspond these pixels across imitation and demonstration via multi-view self-supervised pixel feature detectors. In that sense, we create dynamic point entities consisting of pixels for every time step that we correspond across imitation and demonstration. We detail each one below.

4.2.1 Robust Object-Instance Detection via Synthetic Data Generation

Since we define object entities as any coherent rigid or non-rigid entity in the scene, we can conveniently make use of standard object-detection and segmentation machinery to robustly detect relevant object entities in the scene.

Object detection in general is a well-explored task in the domain of computer vision. There are numerous techniques to detect objects in scene, among which neural architectures like [51] showed superb performance on the detection task. Recently, more advanced architectures such as Mask R-CNN [21] also predict instance-specific masks in parallel to prediction object bounding boxes.

For most manipulation tasks such as in this work, off-the-shelf detectors pre-trained on a generic dataset like MS-COCO are not sufficient for detecting all relevant objects in the scene. This is due to the fact that most objects in these datasets do not cover the entire range of objects that are present in a particular manipulation setup. Even if the object category is part of a category in a human-annotated dataset, we still need to train *instance-specific* detectors in order to reliably track the objects during manipulation (e.g., different instances of the same object class cannot be uniquely identified by the detector if it has not been trained on these specific object instances before, greatly exacerbating the problem of accurately tracking them.) Moreover, in a typical manipulation task, objects often undergo partial occlusion, leading to detection failures if the appearance of the partially occluded object deviates too much from its appearance in the training set.

To alleviate the latter problem, one could create a variety of different occlusion scenarios and generate a custom dataset by hand-annotating all objects in the occluded scene. However, manually generating the masks and bounding boxes for the relevant objects in a variety of configurations to fine-tune the detector is a tedious process. **Instead, we propose *data-dreaming* for learning robust object-instance detectors**, where we create a sufficiently large synthetic dataset on-the-fly with a small set of ground-truth masks of all relevant object categories in the scene.

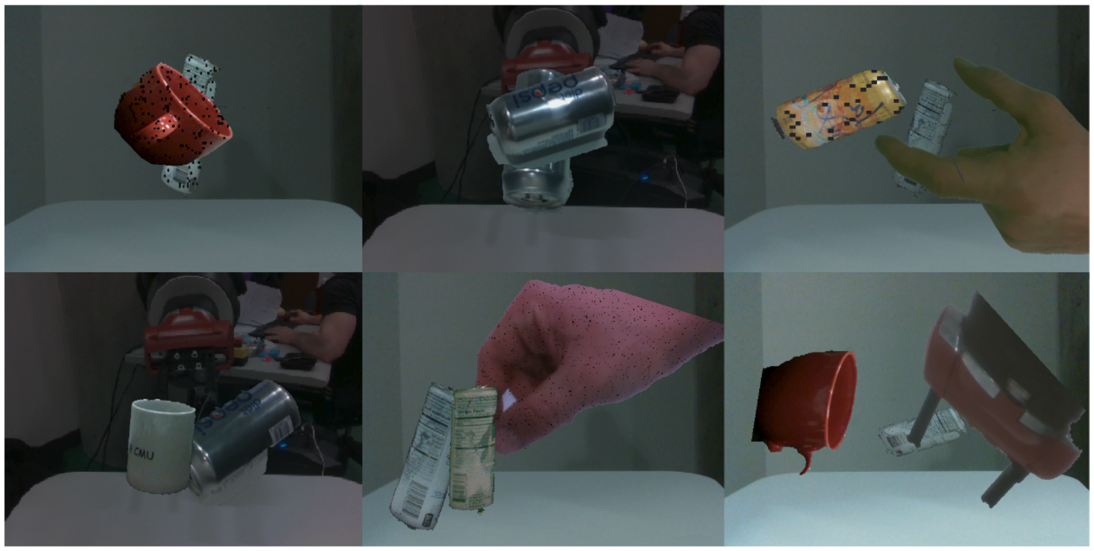

Figure 4.3: Sample images of synthetically created dataset.

Objects' RGB masks are automatically obtained using RGB background subtraction and *GrabCut* [52] for the initial frame, where we assume the objects are not occluding one another, as shown in Figure 4.4.

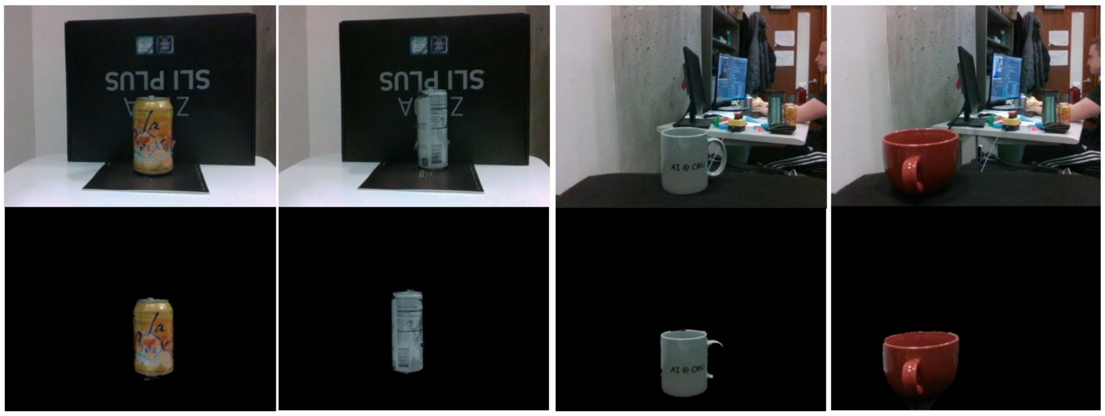

Figure 4.4: Object masks created from unoccluded views.

Each image is also given a label, e.g., *mug*, to identify the relevant object in the scene. We then create a set of training images by randomly overlaying all available object masks from our custom object library. The segmented object masks often partially overlap with each other within those synthetically generated images. Such overlaps help the detector to be robust under partial occlusions, still predicting the correct *amodal* bounding box, i.e. predicting the entire bounding box even if the corresponding object is only partially visible. Since we generate such images, we automatically know the ground-truth pixel bounding box and mask that correspond to each object in each image, as seen in Figure 4.5.

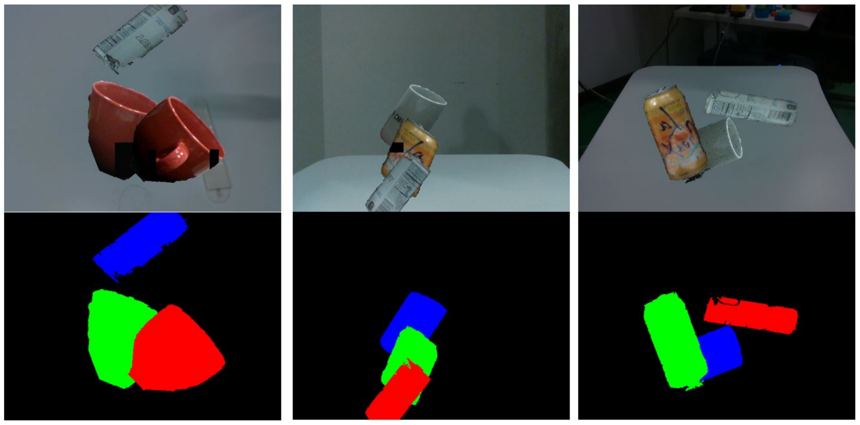
Figure 4.5: Partially occluded object mask automatically created from singe object ground truth object masks.

We then fine-tune a Mask R-CNN object detector [21] to predict boxes and masks for the synthetically generated images. To make the detector more robust, especially towards different lighting conditions and viewpoints, we further apply a set of standard image augmentations to each mask and the background, such as pixel dropout, scaling, rotating, and pixel perturbation. In Figure 4.3, we show a random subset of the training set used for specific tasks in our experiments.

Given multiple objects present in the scene, we need a way to indicate to our agent which ones are relevant for the task. In our case, we provided labels to the objects within the training set, and thus, we can specify the label of the target objects in the scene. The output of our visual detector is thus an ordered set of bounding boxes and segmentation masks that correspond to the set of relevant objects in the scene. Refer to Figure 4.6 for an example output sequence.

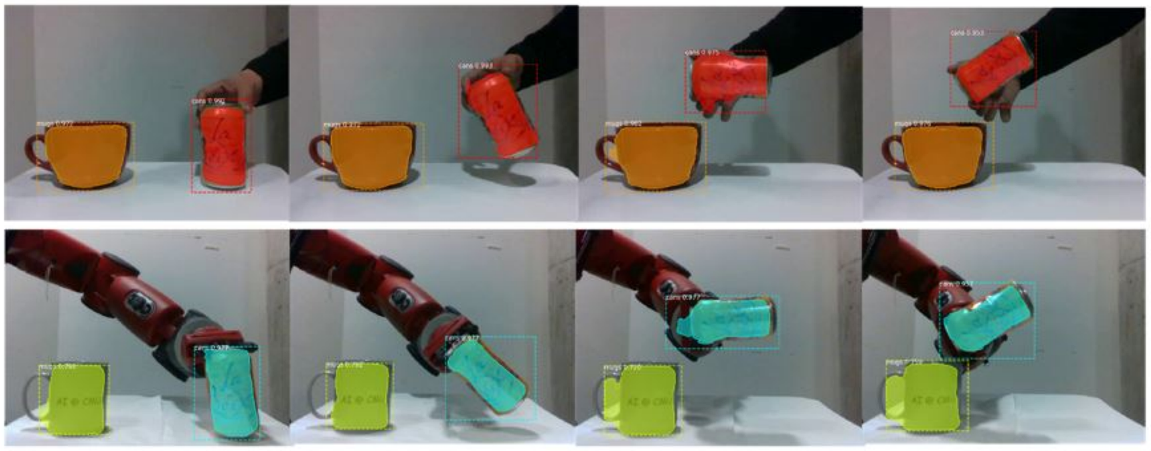
Figure 4.6: Example sequence of detected object entities, *mug* and *can*, during demonstration (top) and imitation (bottom).

4.2.2 Human Hand Detection

End-to-end approaches such as [34] rely on large amounts of data to train hand-to-robot correspondences, and are therefore prohibitive in few-shot learning scenarios. Instead, we opt for state-of-the-art hand detectors of Simon at al. [58] to detect human finger joints, and obtain their 3D locations using a D435 Intel RealSense RGB-D camera. Figure 4.7 shows an example detection of the hand keypoint detector. For the very reason that this human-to-robot mapping is dependent on the specific robot model used as well as the task itself, we decided to establish the separate node class $\mathcal{V}_h$ for hand/robot entities to facilitate task-dependent modelling without harming the generalizability of defining generic object nodes $\mathcal{V}_o$.

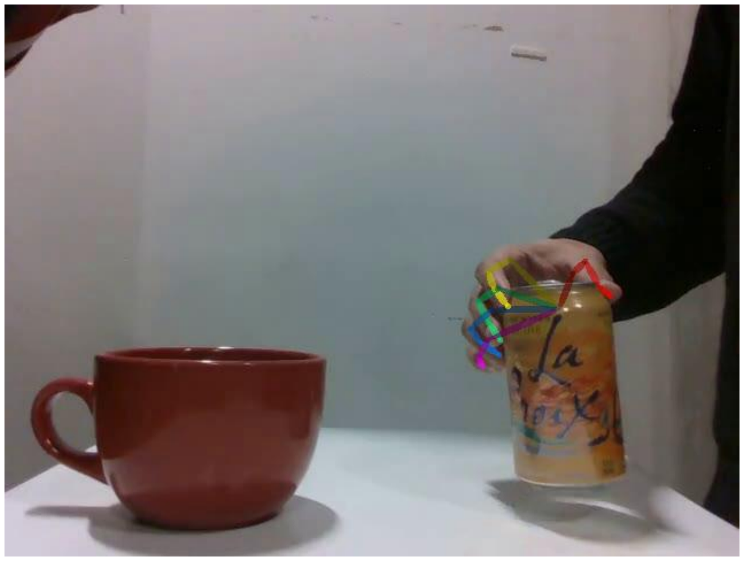
Figure 4.7: Example detection of *OpenPose* hand keypoint detector.

We map the fingertips of a Baxter robot's parallel-jaw gripper to the demonstrator's thumb and index finger tips. We detect grasp and release actions by thresholding the distance between the two fingertips of the human during the demonstration of the task. Locations in 3D of the tips of the robot's end-effector are computed using forward kinematics and a calibrated camera with respect to the robot coordinate frame.

4.2.3 Multi-View Self-Supervised Pixel Feature Learning

Our goal is to learn feature descriptors of pixels that are robust to changes in the object pose or camera viewpoint. An agent that can alter its camera view at will observes a static scene from multiple views, and, through 3D reconstruction and triangulation, infers visual correspondences across views [17]. These self-generated correspondences drive visual metric learning to recover such correspondences in the absence of viewpoint information, namely, to learn deep feature descriptors of points that are robust to changes in the object pose or camera viewpoint. We collect videos of the scene containing objects of interest in a fully autonomous manner. We use an RGB-D camera attached to the robot's end-effector and move the camera while following random trajectories that cover many viewpoints of the scene and at various distances from the objects. We use the robot's forward kinematics model to estimate the camera poses via hand-eye calibration, which, in combination with the known intrinsic parameters and aligned depth images, allows for robust 3D-reconstruction of the scene, and provides accurate pixel correspondences across different viewpoints. The complete feature learning setup is illustrated in Figure 4.2. During training, we randomly sample image pairs from the video, and generate a number of matching and non-matching pixel pairs. We then minimize pixel-wise contrastive loss [11, 56], which forces matching pixels to be close in the learned feature spaces, maintaining a distance margin for non-matching pixels. This pixel feature learning pipeline produces a mapping from an RGB image to dense per-pixel descriptors, as illustrated in Figure 4.8.

Note that such metric learning for obtaining pixel features is performed given only within-instance correspondences, but within each class, generalization across different objects with different geometries and textures comes freely since learning such embedding are expected due the limited VC dimension of the model. This enables our VEG representation with powerful generalizability to learn skills even with objects unseen in the demonstration. In Figure 4.9, we can see exemplary visualizations of such detected correspondences.

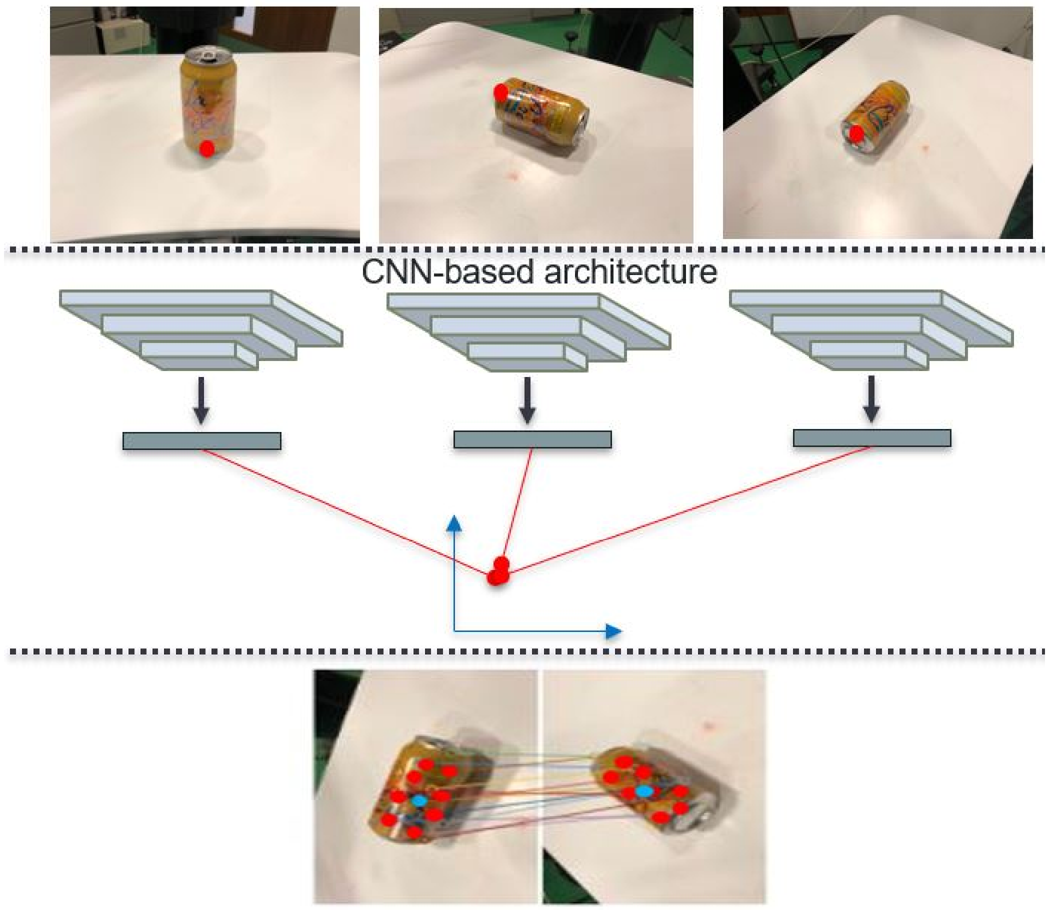

Figure 4.8: Metric-based learning of dense pixel features.

A problem we encounter is that we cannot robustly track specific pixels over time as we are doing for the object centroids because the closest neighbor of a pixel in the imitation video is sometimes not correct, as seen in Figure 4.9 in the bottom row. This prevents us from using these pixels as time-consistent point entities. Recent approaches such as [39] would make an ideal candidate for dealing with this problem, but are currently limited by hand-designing object-specific keypoints. For the scope of this work, we stay within the class of rigid objects, and hence, we assume that relative spatial arrangements amongst all pixels belonging to the same object stay constant. In that sense, matching relative spatial arrangements between pixels across imitation and demonstration boils down to matching the pose of a rigid object. We then proceed by sampling enough pixels such that the majority of them will have sufficiently precise correspondences to outweigh the spurious correspondences. Note that this approach will fail if we introduce non-rigid objects, but using sparse pixel features such as [39] would alleviate this shortcoming. More formally, we sample a fixed number of pixels dynamically for every time step over the segmentation mask of the object within the demonstration video, and query the corresponding closest neighbor in the imitation video for the corresponding associated detection of the object. The overall 3D-distance is then computed via summing over all pair-wise distances for each pixel feature pair with respect to the object centroid in the demonstration and imitation for the current time step via $\frac{1}{N}\sum_{i=1}^{N}\left\|\left(\mathbf{x}_D^{t,i}-\mathbf{x}_D^{t,centroid}\right)-\left(\mathbf{x}_I^{t,i}-\mathbf{x}_I^{t,centroid}\right)\right\|$, where N denotes the number of sampled pixels. This summation is exactly equal to the summation over edges connecting point entities $\mathcal{V}_p$ with their respective object entity $\mathcal{V}_o$, as described in 4.1.

4.3 Motion Attention for Dynamic Graph Construction

We consider edges, i.e. $\mathrm{att}(\mathcal{E}^{t,(i,j)}) = 1$, for all edges between the anchor object node and all other object nodes in the scene, including the hand/robot node, and all point nodes that belong to the anchor object. The anchor object is any object in motion, and in the case of no moving objects, (*e.g.* the hand performing a reaching task), the next moving object in the future is identified to be the anchor object. Such anchor-centric graph structure allows flexible camera viewpoint and is robust against shift in absolution spatial configurations of the objects.
This type of motion attention, or *motion saliency*, as it is also referred to, is a well-established principle that drives human attention during imitation. [31] use AR markers to estimate such task relevance based on motion saliency. In our framework, we can simply use our object detection network to track objects in time to infer movement between the different objects in the scene. In our framework, we can simply use our object detection network to track objects in time to infer movement between the different objects in the scene. This dynamic motion attention is illustrated in Figure 4.1.

4.4 Policy Learning

Our goal is for the robot to learn a policy that successfully imitates the a task given by a human via a video demonstration. We formulate this as a reinforcement learning problem, where at each time step the cost is computed as in Equation 4.1. We use behavioral cloning to infer opening

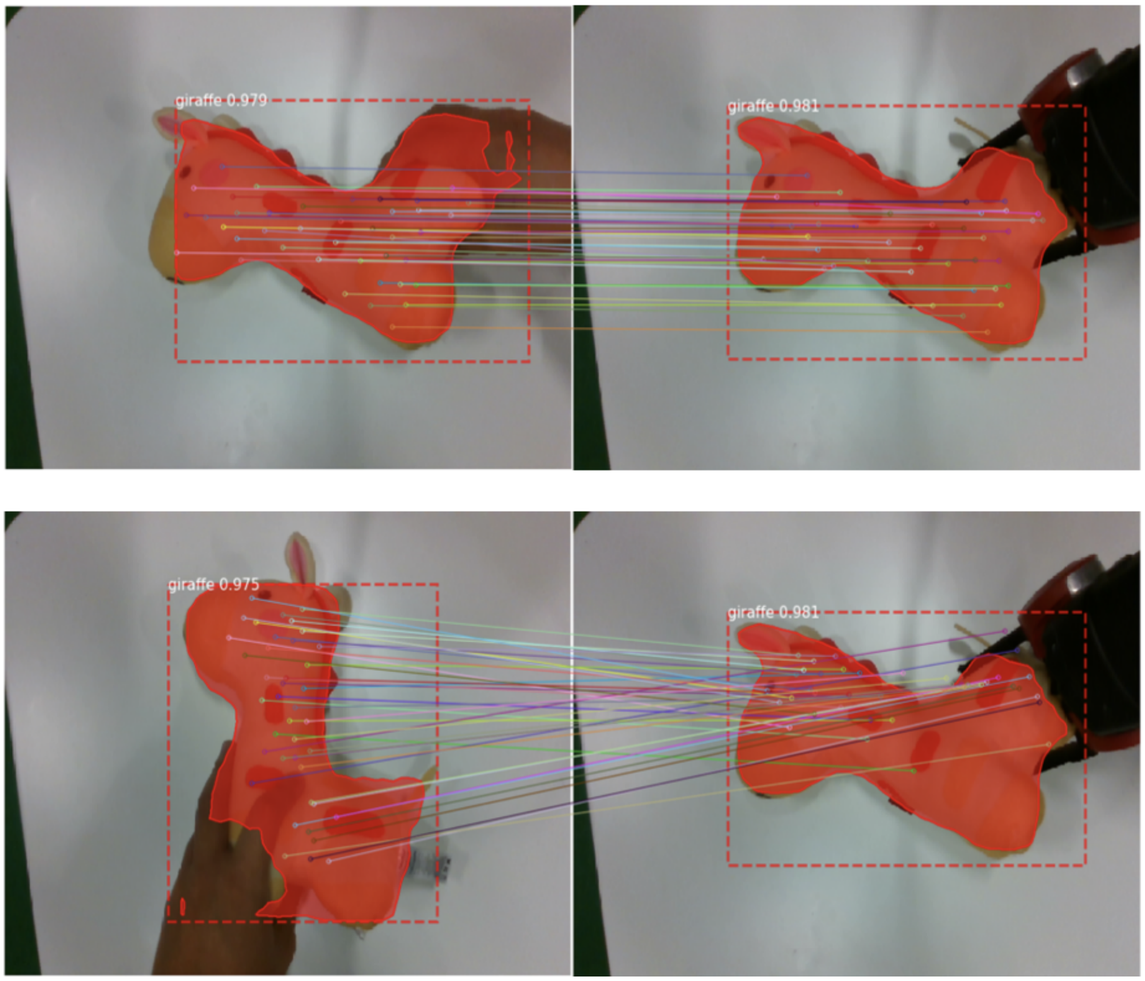

Figure 4.9: Example correspondences for pixel feature detection

and closing of the robot gripper by thresholding the distance between the human index finger and thumb during the demonstration.

We use PILQR [10] to minimize the cost function in Eq. 4.1, a state-of-the-art trajectory optimization method that combines a model-based linear quadratic regulator approach with path integral policy search's ability to handle non-linear dynamics, which are scenarios that vanilla LQR-based approaches usually struggle with. We learn a time-dependent policy $\pi(\mathbf{u}_t|\mathbf{x}_t;\theta) = \mathcal{N}(\mathbf{K}_t\mathbf{x}_t + \mathbf{k}_t, \mathbf{\Sigma}_t)$, where the time-dependent control gains are learned by alternating model-based and model-free updates, where the dynamical model $p(\mathbf{x}_t|, \mathbf{x}_{t-1}, \mathbf{u}_t)$ of the *a priori* unknown dynamics is learned during training time. The actions $\mathbf{u}_t$ are defined as the robot end-effector's 3D velocity and its rotational velocity about its end-effector principle axis, giving a 4-dimensional action space. In tasks where we also learn how to grasp, we further include the robots gripper state as a binary action value (open/close), giving a 5-dimensional action space.

The input to the policy consists of the joint angles, the 3-dimensional end-effector (X, Y, Z)-position, and the current graph configuration of the scene, concatenated into one state vector. We note that this state space is over-informative in the sense that the end-effector position can be determined by knowing the joint positions only, but we found the learning to be more stable when including the end-effector 3D location.

While the cost is computed as the overall sum of all distances between all pixel features that belong to one object, using every single pixel feature as part of the state representation would lead to a prohibitively large state space. Apart from this fact, our pixel features are created in a dynamic fashion and re-sampled at every time step, ruling them out to be used as part of the effective state space which requires dynamic consistency in time. Nonetheless, we can take the average distance between the demonstrator and learner as a state space, which yields a 1-dimensional feature for each node, i.e. $dim(\phi_i)) = 1$.

For N objects, this featurization scheme results in a $3 + n_{joints} + \sum_{i=1}^{N_{anchors}} (N_{total} - 1) * 3 + \sum_{i=1}^{N_{anchors}} dim(\phi)$ dimensional state space, where n_{joints} denotes the number of joints of the robot and ϕ encodes the chosen node feature for a visual entity in the graph. ϕ allows for adjusting the state representation without being tied down to the specific cost formulation 4.1. As mentioned above, in our case $dim(\phi_i) = 1$. $N_{anchors}$ denotes the number of anchor objects during a demonstration, and N_{total} denotes the overall number of objects in the scene. Since the state space has to be consistent throughout an entire episode, we thus only include relative arrangements and pixel features into our state space of objects that are at some point in time during the demonstration labelled as an anchor, while dynamically adjusting the cost term depending on when exactly the objects are labelled anchors. For example, the pixel feature and all relative arrangements with respect to an object that is only moving during the second half of an episode will be part of the state space for the entire episode, but its pixel feature and the relative arrangements to other objects will only be included in the cost during that second half.

4.5 Implementation Details

Robotic System Platform: We conducted our experiments on a Baxter robot from Rethink Robotics using ROS [48] as the development system to enable communication between the different system components. For grasping, we use Baxter's parallel-jaw two-finger gripper. For visual processing, we use Intel RealSense D435 RGB-D cameras, which use stereo vision to estimate depth. We use a 640x480 resolution for both the RGB and the depth input. Human involvement was limited to collecting one demonstration for each task as well as manually resetting relevant objects in the scene if their end configuration differed from their start configuration. For the control of the robot, we implemented both Cartesian task-space position and velocity control for the left arm of Baxter. While the velocity control allowed for smoother control outputs, the estimation of the Jacobian was often not accurate enough to provide accurate task-space control near singular configurations. In general, Cartesian velocity control yielded much smoother trajectories, which is why we opted for this controller for our experiments.

We placed the D435 camera on the end of the right arm of the robot, and calibrated the camera with respect to the robot's base frame using standard hand-eye calibration. The calibration was needed to automatize the data collection for the pixel feature learning which requires known extrinsics of all training views and to infer the robot's end-effector position with respect to the camera frame. The complete system setup can be seen in Figure 4.10. We statically mounted another D435 camera on the robot's torso to cover more viewpoints which the hand-mounted camera cannot cover. However, for most of our experiments using the hand-mounted camera's viewpoints sufficed.

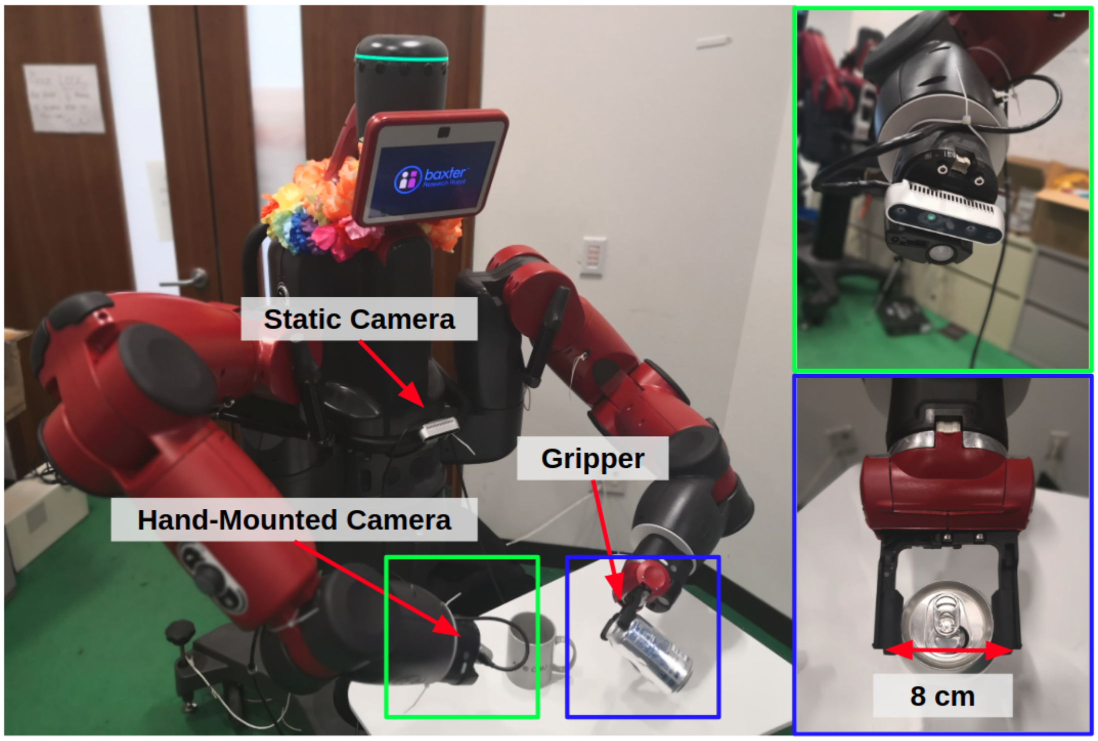

Figure 4.10: Robotic system setup.

For all experiments, we only allow for movements in (x, y, z)-direction, locking all rotational degrees of freedom. For both the pushing and the pouring experiments we unlock allow the end-effector to rotate around its principal axis. We restrict movement in z-direction for the pushing experiments to avoid unnecessary collision with the table. Using the task space velocity commanded by the policy, we compute the required command in joint velocities via $\dot{\mathbf{q}} = \mathbf{J}^{\dagger}\dot{\mathbf{x}}$. Here, $\mathbf{J}^{\dagger}$ is the Moore-Penrose pseudoinverse of the robot's Jacobian defined by $\mathbf{J}^{\dagger} = (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T$. We use an open-source kinematics and dynamics library, *Baxter PyKDL*, to obtain the robot's Jacobian. We note that estimating the Jacobian for every time step led to unexpected instabilities in the policy learning process. For that reason, we computed the Jacobian once in the beginning, and used this Jacobian as an approximation for all subsequent time steps. In some sense, the deviation of this approximation from the actual Jacobian is now subsumed into the environment dynamics learned by the policy.

Synthetic Data Generation: We obtained the object masks by using an off-the-shelf implementation of *GrabCut* available in the OpenCV Python API after applying standard background subtraction in the RGB space. Depending on the object, the background subtraction procedure required fine-tuning of the RGB threshold. We augmented the obtained object masks with a variety of different augmentations using the *image augmentation* open-source Python library. More specifically, we applied affine transforms, including shear, translation, rotation, and scaling. We also applied static transforms, including pixel intensity perturbation, Gaussian noise, and pixel dropout. Especially the latter turned out to be crucial for training detectors robust to different lighting conditions. The *can* object detectors in particular were not robust due to their reflective surface: Since the training set included views of the cans with specific reflections of lighting dependent on the location of the can as well as the lighting conditions in general, the detector is trained only on this very specific subset of all lighting conditions. Creating enough variation in this regard would amount to a tedious process of collecting additional data. However, using pixel dropout also led to big improvements by emulating reflection artifacts.

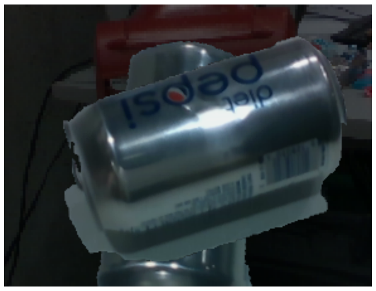

Figure 4.11: Example of reflective appearance of the *can* object class.

We created the synthetic dataset by randomly overlaying between 1 and 5 images on each background. Overlaying too many images resulted in highly fragmented object appearances that made it hard for the network to learn higher-level features (a too small crop off of different objects looks too similar even across different object instances). On the other hand, overlaying too few images did not create a sufficiently rich enough dataset of partially occluded object instances, defying the purpose of this robustification process. An ideal dataset would contain both unoccluded object instances as well as a diverse set of partial occlusions of a sufficient size. We found that setting the number of overlaying images to between 1 and 5 was sufficient, and we additionally encouraged object masks to be placed in the center of the image, in addition to the random augmentation created through affine translations. One should also ensure to only select backgrounds that do not contain any object instance of the newly labelled object classes. For example, using a background with bottles visible in the scene is counterproductive towards training a bottle detector since it would then use these ”background bottles” as negative examples.

Object-Instance Segmentation: We use a publically available implementation of Mask R-CNN by Matterport. It implements the standard Mask R-CNN architecture according to [21]. We use a *ResNet101* backbone architecture, and a learning rate of 0.0001 to fine-tune a model that was pretrained on the MS-COCO dataset. We experiment with both training a single model across all objects of all tasks, but achieved much better detection performance when training separate models for different tasks. This also allows for more targeted fine-tuning of the model and adaptation of the dataset. For example, when making the training more robust towards different lighting conditions as mentioned before, fine-tuning a model that was exclusively trained on cans, mugs, hands and the robot end-effector would not harm the performance of a model that was trained on the shapes used in other tasks.

Pixel Feature Learning: We build on top of the open-source code of [17]. We build our custom data collection pipeline, which automatically collects a training data of a given object. The robot arm follows a kinesthetically taught trajectory, during which it takes images with the D435 camera mounted to its end-effector. We then use the same synthetic dataset augmentation procedure as for training the Mask R-CNN to robustify the learning process of the pixel features. We noticed that the absence of this augmentation process led to highly volatile features that would arbitrarily deteriorate in performance during policy learning when a current viewpoint would deviate even slightly from the viewpoints seen during the feature learning process.

Policy Learning: We make use of the open-source *Guided Policy Search* Python library [15] for policy learning. More specifically, we implemented our custom Baxter interface to make use of their implementation of the rollout-based policy learning methods *PILQR*. We also experimented suing PI^2: while this also led to convergent policies, it took longer to do so and the convergent policy was more fragile than the one learned with *PILQR*. The policy learning process required a great amount of hyperparameter tuning. Among others, the following hyperparameters proved to be the most crucial ones in ensuring reliable convergence of the policy learning process:

1. Gains for the chosen action parametrization, in our case task space velocities and end-effector rotation delta
2. Weights for each cost term, i.e. the different edge weights of our main cost function, including a generic action cost term
3. Number of rollouts before for each policy parameter update cycle
4. Number of time steps during each episode
5. Length of each time step (constraint by inference time of the entire pipeline)

For more detailed information regarding the hyperparameter setup refer to Appendix A.1.

Chapter 5

Experiments & Results

We test our visual imitation framework on a Baxter robot, and consider the following tasks to imitate:

1. **Pushing**: The robot needs to push an object while following specific trajectories.
2. **Stacking**: The robot needs to pick up an object, put it on top of another one and release the object.
3. **Pouring**: The robot needs to pour liquid from one container into another.
4. **Transportation**: The robot needs to transport, i.e., rotate and translate, a pen into a target configuration.

Examples for each of the tasks are shown in Figures 5.1, 5.3, and 5.4. For every task, we train corresponding object detectors using synthetic data augmentation and point features using multi-view feature learning. Note that both processes are fully automated and do not require human demonstrations or robot interactions, rather than simply a camera that is setup to move around the scene in a prerecorded fashion.

Our experiments aim to answer the following questions:

- Is the proposed VEG based cost function discriminative enough for successful skill imitation?
- How does our VEG encoding compare against convolutional image encodings of previous works [57]?
- How robust are VEGs to variability in the objects' spatial configurations and background clutter?
- How well can VEGs generalize across objects with different shapes and textures?

Baseline: We compare our method against time-contrastive networks (TCN) of Sermanet et al. [57]. We are not aware of other instrumentation-free methods that have attempted single shot imitation of manipulation skills, without assuming extensive pre-training with random actions for model learning [43]. TCN trains an image embedding network using a triplet ranking loss [24] ensuring that temporally near pairs of frames are closer to one another in embedding space than any temporally distant pairs of frames.

In this way, the feature extractor focuses on the *dynamic* part of the demonstration video. We implemented the TCN baseline using the same architecture as in [57], which consists of an Inception-v3 model [61], pretrained on ImageNet, up to the Mixed 5d layer which is then followed by two convolutional layers, a spatial softmax layer and a fully-connected layer. The network finally outputs a 32-dimensional embedding vector for the input image. For each imitation task, we train a corresponding TCN using ten video sequences, five human demonstrations and five robot executions while performing tasks with relevant objects and environment configuration, together with the human demonstration we provide for learning the policy with VEG . Our method does not require additional data as TCN does.

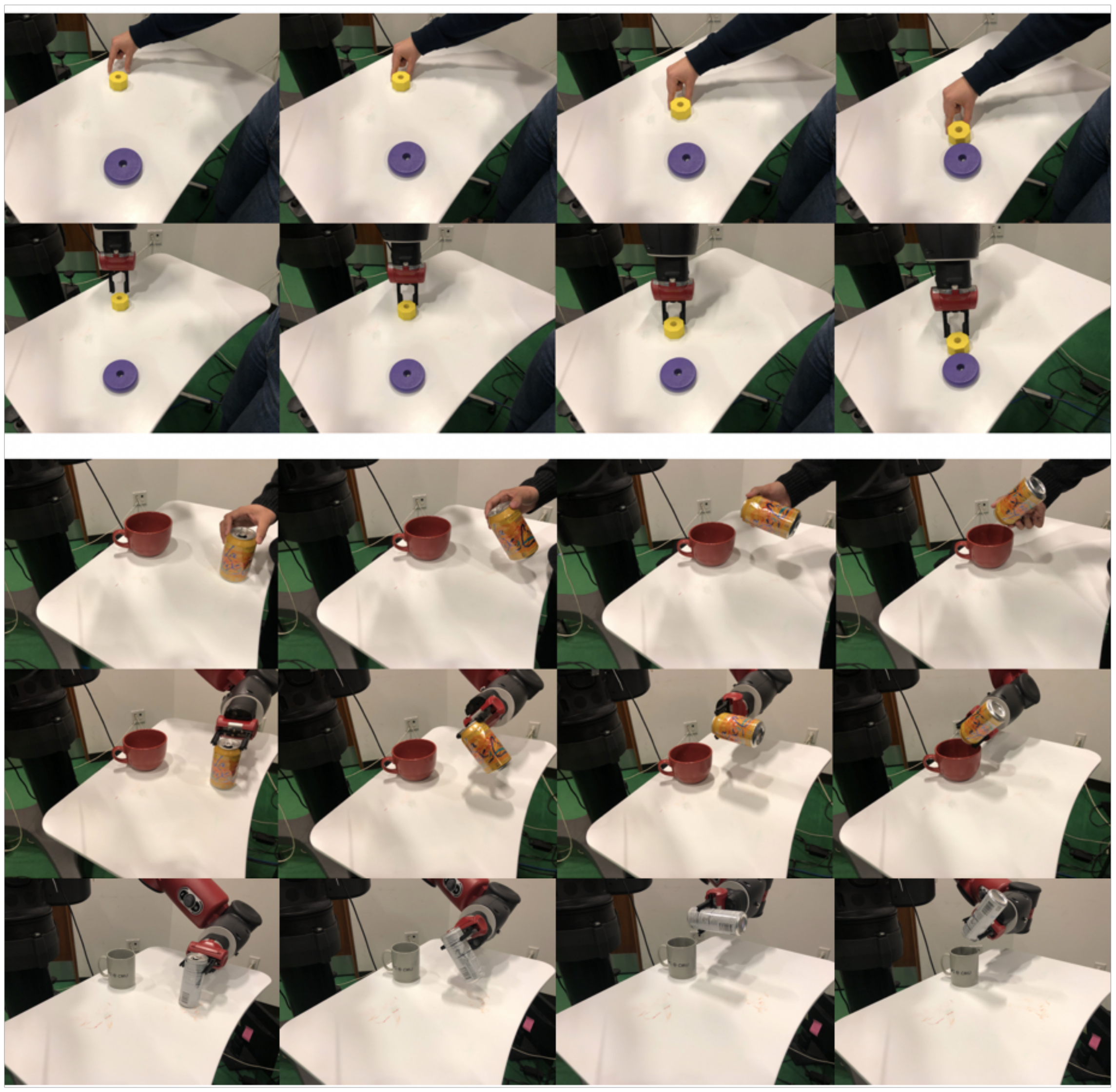

Figure 5.1: **Examples of the imitation tasks we consider.** Top: *pushing*. Bottom: *pouring* demonstration and imitations with different objects. Example of the *stacking* and transportation task are illustrated in Figures 5.3, and 5.4, respectively.

5.1 Cost Shaping

We evaluate the robustness of the proposed VEG-based cost function in Eq. 4.1 to background clutter and its discriminativeness in guiding correct visual imitation. In Figure 5.3, we consider a human demonstration of a stacking task and various robot executions under different conditions. We then show a comparison of the corresponding cost curves for each robot execution with the human demonstration, computed by the proposed VEG based cost function and the TCN [57] embedding cost, where we normalize each reward by the maximum absolute value of each respective cost. The horizontal axis denotes time and the vertical axis denotes imitation cost. The proposed graph-based cost correctly identifies all correct executions, despite heavy background clutter in the fifth row, and correctly signals the wrong imitation segments in second and third rows. In contrast, the TCN cost curves are largely arbitrary. Cost curves which are highly discriminative in case of both successful imitation and also failure cases are critical for effective policy learning, which we discuss in the following section.

5.2 Policy Learning

Implementation Details: For each task, we provide the robot agent with a single human demonstration to imitate. We run the experiment on a NVIDIA GTX 1080 Ti GPU. The inference time of the entire VEG pipeline is approximately 0.4s, allowing real-time position & velocity control. A fixed episode length is adopted for all experiments. We use an existing implementation of PILQR [15] and sample a fixed number of trajectory rollouts per iteration before each policy update in training. The action space of the policy includes translational movement along x, y, and z axes in the world frame, rotation of the robot's end-effector and a binary grasp/release action, depending on the task. For specific hyperparameter values refer to Appendix A.1. Note that both *pushing* and *stacking* only require translational movement of objects, so we compute cost only based on the *object* level nodes in our graph. The *pixel* level nodes are enabled for the *pouring* and *transportation* tasks, since rotation is critical for these.

We employ a simple form of tracking-by-detection: we only allow one specific instance of each object class to be contained within the scene, i.e., we take the highest scoring detection of each relevant object, based on the thresholded confidence score. If no object instance is detected for a given frame during imitation, the last known detection is used for cost computation.

Furthermore, we smooth the demonstration hand keypoint trajectory via *Savitzky-Golay* filtering with fixed window size and polynomial order since the raw keypoint trajectory is noisy. We also smooth the pixel feature values, i.e., the average distance deviations between demonstration and imitation pixel features, using a simple moving average filter of window size 3. Without such filtering, the policy learning diverges. We note that we do not employ any kind of advanced pre-processing such as state space normalization or whitening the data, and only utilize basic filtering as mentioned above.

5.2.1 Pushing

For imitating pushing, the robot needs to push the yellow octagon towards the purple ring following the trajectory showed by the demonstrator (Figure 5.1). The task is considered solved if the octagon's distance to the ring deviates less than 1cm from the final distance in the demonstration. We evaluate three task variations:

- Pushing the object following a straight line.
- Moving the yellow octagon along a straight line with it being always grasped.
- Pushing the yellow octagon along a trajectory with a sharp direction change of 90 degrees. Imitating such direction change requires the robot to change the point of contact with the object during pushing.

Task	object	VEG	TCN
2*straight-line	yellow octagon	5/5	1/5
	orange square	4/5	0/5
straight-line-grasped	yellow octagon	Success	Failure
direction-change	yellow octagon	Success	Failure

Table 5.1: Success rates for imitating pushing tasks under different settings.

For straight-line pushing, we attempted imitation in two different environments, one with a yellow octagon block, and one with a smaller orange square block to evaluate the generalization capability of our graph-based framework across objects with different geometries. We essentially use the object detector for the small orange square block in the second case, in place of the (self-supervised) detector of the yellow octagon block.

In order to test the robustness of VEG to variations in the objects' spatial configurations for the basic straight-line pushing task, we randomly perturb the starting configurations of all the objects and the robot's end-effector within a norm ball of diameter 6cm, and run the policy training 5 times for each object. With the VEG representation, the robot's policy converges after 6 iterations in all tasks. We consider the task solved if the robot is able to push the object to within 1cm of the desired target position. We report the success rate for each case in Table 5.1. The robot successfully solves the task for all 5 runs for the original yellow octagon, and for 4 runs out of 5 for the smaller orange square, demonstrating robustness against perturbation in objects' spatial configuration and strong generalization over objects with novel geometries. TCN failed in almost all runs, and for simple straight-line pushing could not reach even a quarter to the target even with significantly more iterations.

For imitating straight-line-grasped, unlike the demonstrator, the robot is forced to keep the gripper closed and thus is only allowed to push the object without grasping it. To solve this task, the robot cannot strictly imitate the hand trajectory exactly, but should only use it as a weak guidance signal to push the object along the desired trajectory.

The robot solves both straight-line-grasped and direction-change pushing after eight iterations of trajectory optimization. The cost plot for direction-change is shown in Figure 5.2 as an example. TCN fails to solve any of the tasks, and in general performs very poorly in tasks where the robot does not have continuous contact with the object.

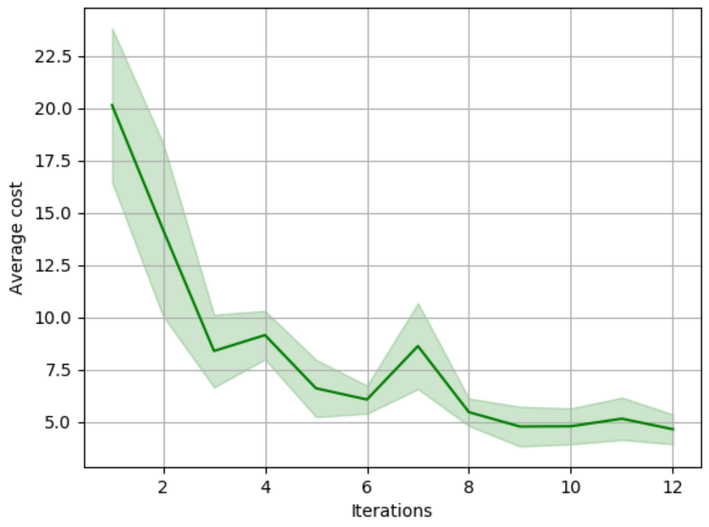

Figure 5.2: **Training cost curve for *pushing* task with direction-change.** We show the mean and standard variance of the average cost over all rollouts for a given iteration, where the average cost is obtained by averaging over an entire episode. The policy learning converges after 8 iterations, and learns a skill to solve the task successfully. Note that policy learning took longer than in the other tasks (usually 3-6 iterations) due to the difficulty of the abrupt direction-change.

5.2.2 Stacking

In this task, the robot needs to reach and grasp the yellow octagon, position it on top of the purple ring, then release it, as shown in the 1st and 4th rows in Figure 5.3. The task is considered solved if the octagon is successfully placed on top of the ring. As in our pushing experiments, we randomize the starting configurations of the scene and the robot, and evaluate our method on this augmented setup for stacking both yellow octagon and the orange square. We report the results in Table 5.2. VEGs enables robust policy learning for the yellow octagon, and shows generalization towards objects with novel geometries that have not been observed during demonstration. Being flatter than the octagon, the square is much harder to grasp for the robot.

The TCN baseline performs poorly because its use of image-level embedding is not capable of learning a successful grasping action, which requires precise and coordinated end-effector movement. We then evaluate TCN with a simpler task: stacking an already-grasped yellow octagon (simple-stacking in Table 5.2). The TCN performs reasonably well on this task, demonstrating its ability to imitate smooth trajectories but incapability of learning fine-grained grasping actions.

5.2.3 Pouring

For the pouring task, as shown in Figure 5.1, the robot needs to simultaneously translate and rotate the yellow can to reach the desired orientation above the mug. The imitation is considered

Task	object	VEG	TCN
2*stacking	yellow octagon	4/5	0/5
	orange square	3/5	0/5
simple-stacking	yellow octagon	5/5	3/5

Table 5.2: Success rate for stacking with different starting configurations

successful if the robot pours a non-negligible amount of liquid into the mug. Using the same pouring demonstration, we additionally evaluate generalizability of VEGs by training with using a novel object with a different shape and distinct textures.

Trajectory optimization converges after ten iterations, and solves the task successfully for all the objects. The TCN baseline is able to move the can along the trajectory, but fails to rotate it to the right configuration for successful pouring. Videos of the robot imitating pouring with object generalization are included in the supplementary materials.

5.2.4 Transportation

For this task as shown in Figure 5.4, the robot needs to transport pens to a final goal location as indicated during the demonstration. This task serves the purpose of showing how even fine-grained manipulation tasks can be imitated in a reliable manner, even across different object instances. The task is considered solved if the front of the pen crosses the imaginary line defined by passing through the yellow ring and being parallel to the long end of the table. We evaluate generalizability of VEGs by additionally training a policy with different pens than used during demonstration. The green pen has been seen during pixel feature training, while the thinner purple pen has not been seen during feature training, Trajectory optimization converges after three iterations in this scenario.

5.3 Discussion

Imitating the human hand critically improves performance when imitating the reaching and grasping motions in human demonstrations, which cannot be easily inferred from any object motion.

Our method relies on unoccluded areas of the scene, and there is no current attempt to keep track of occluded entities. Following this, a clear avenue for future work is learning to track through occlusions, and use a temporal visual memory to extract the reward graph from, as opposed to relying on current observation. Active vision could also be used both to undo occlusions during imitation, but also to observe the demonstrator from the most convenient for the imitator's viewpoint. An interesting avenue for future work would be learning a model over motion of visual entities, and use it as a prior for better exploration during policy search.

As for comparing with the TCN baseline, we note that we tried to improve the policy learning by providing more diverse demonstrations for the TCN feature learning, which, however, led to worse results during the policy learning. We concluded that overfitting the features by providing specific demonstrations resemblant of the actual tasks as we did for the initial demonstration set led to better features than a more diverse demonstration set. This makes sense since we only focus on hard-imitation of a given visual demonstration rather than generalizing over diverse initial conditions.

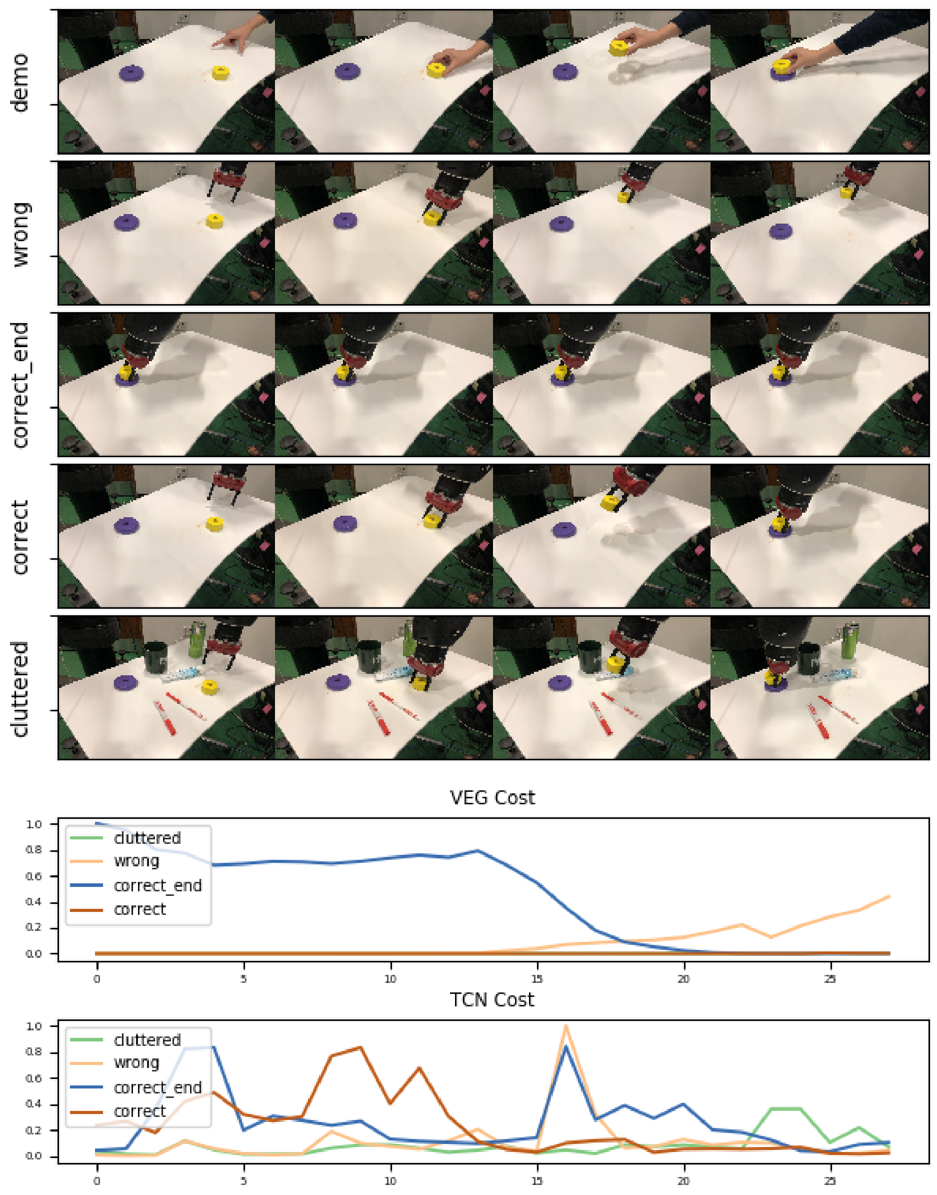

Figure 5.3: **Cost comparison of the proposed VEG cost function and the TCN embedding cost [57].** The VEG based cost curves are highly discriminative of the imitation quality for the stacking task, in contrast to TCN. Costs are scaled between $[0, 1]$ by dividing by the maximum absolute value of the respective feature space for visualization purposes. The *correct_end* sequence is a simple repetition of the last frame of a successful imitation, showing our loss function serves well as an attractor towards the final goal configuration.

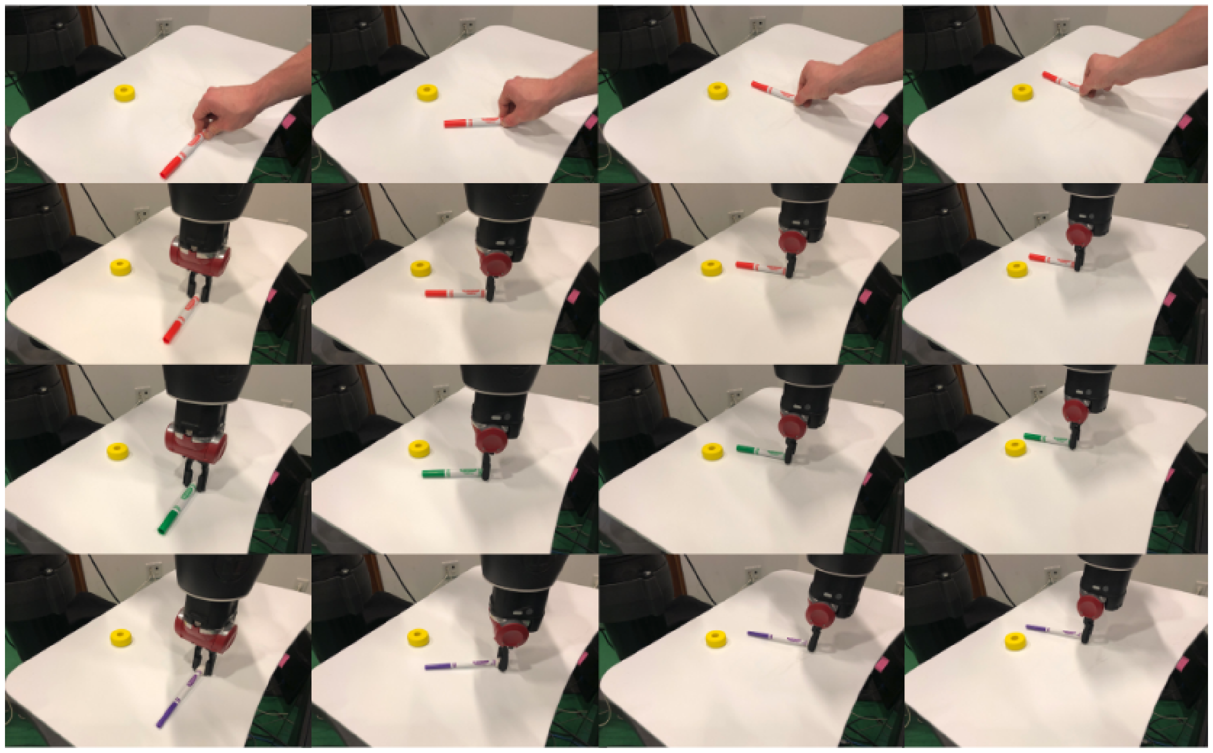
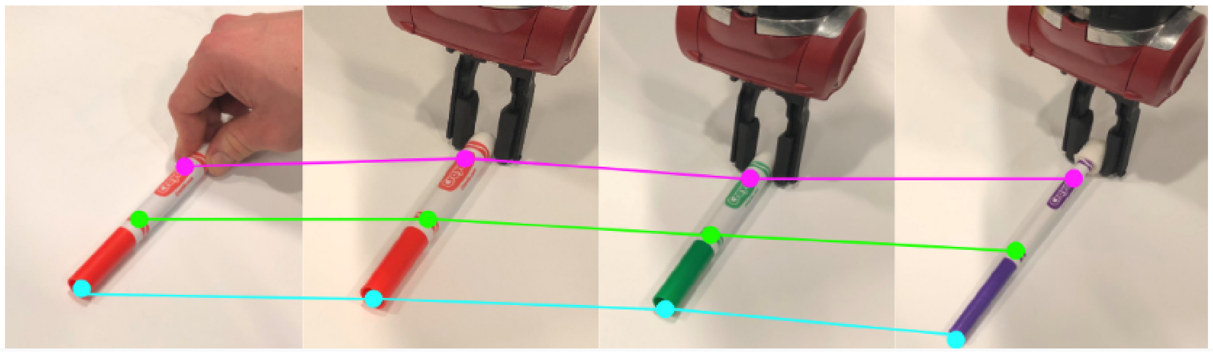
Figure 5.4: **Illustration of feature generalization for the *transportation* task.** Top: (1) demonstration (2) imitation with same pen (3) imitation with different pen that was seen during training (4) imitation with different pen that was **not** seen during training. Bottom: sampled subset of point features for this task and feature correspondence across different object instances.

Chapter 6

Conclusion

We proposed a general framework for visual imitation learning of manipulation tasks given a single human demonstration. We framed the problem of imitation learning as a reinforcement learning problem, more specifically as a trajectory optimization problem.

Within our framework, we learn state representations for visual imitation with a structural bias regarding object attention. We encode images in terms of visual entities and their spatial relationships, and use it to compute a perceptual cost function for visual imitation. Our agent fixates on object- and pixel-level correspondences, and uses synthetic data generation as well as multi-view associations to learn detectors for such fixations. These detectors find corresponding visual entities between the demonstrator and imitator videos. The agreement of the spatial arrangement of visual entities in each frame between demonstrator and imitator serves as a cost function for rollout-based trajectory optimization of the demonstrated skill.

Our experiments suggest that background subtraction, scene-conditioned synthetic data dreaming, and multi-view self-supervised visual learning result in data-efficient learning of detectors for a number of useful visual entities. We show how our proposed framework learns a variety of different tasks from just a single demonstration. This kind of performance was not achieved with plain convolutional image encodings operating on the entire input image, given the same amount of data. Especially in real-life scenarios, collecting annotated data that involve human efforts is a tedious process. Our one-shot imitation learning approach aims to make the most out of a single provided human demonstration.

Between end-to-end learning and engineered representations, we combine the best of both by incorporating important inductive biases regarding object fixation and motion attention in our robot learner. Experimental results on a real robotic platform suggest the generality and flexibility of our approach. Quoting the authors of [8], "Just as biology uses nature and nurture cooperatively, we reject the false choice between hand-engineering and end-to-end learning, and instead advocate for an approach which benefits from their complementary strengths."

6.1 Assumptions & Limitations

For all our experiments, we made the following assumptions:

1. The centroid of each object can be reliably tracked to some extent. This means that while our method is robust to spurious detections or full occlusions for a few time steps, it will generally fail if the object is not detected for too many consecutive time steps.
2. Due to the choice of our point entity detection & correspondence method, we currently cannot handle non-rigid objects.
3. We only use one single human video demonstration for each task.
4. We assume that we can easily generate unoccluded object masks for each relevant object in the scene.

Now we list several limitations of our method along with related potential improvements for future work:

1. With respect to point entities, we only consider edges between the point entities and the object entity they belong to, forming a *star*. In theory, this would still work with non-rigid objects for limited deformation during the demonstration. However, considering edges between point entities within in object would allow for more precise learning of deformations during a demonstration.
2. We do not consider edges between point entities of different objects. This would be especially useful if a manipulation task involves precise interaction between two parts of objects, e.g., when inserting a peg into a hole. Considering an edge between the tip of the peg and the hole, this task could be learned much more precisely by focusing on the interaction between those specific parts.
3. The weights for different types of edges, i.e., hand, object, and points, are currently manually fixed and not learned. Furthermore, the binary attention for the edges is heuristically inferred via motion attention. Meta-learning could be used to learn both the weights and the binary attention to allow for more flexibility in the learning process.

6.2 Future Work

There are several directions to improve on our current method for future work. As mentioned before, we cannot handle full object occlusions, and we rely on tracking-by-detection rather than employing any type of more sophisticated tracking. While the latter could be integrated in a rather straightforward manner, the former is a conceptually much harder problem. In principle, we make use of 2.5-dimensional input data, i.e. we use single-view RGB images with an additional depth channel during inference. Using some type of holistic 3D representation, for example via learning a 3D deep feature representation of the scene using multiple views, one could **deal with full-object occlusions in a principled manner in combination with more advanced tracking**.

In terms of policy learning, there are two directions for improving the generalization capabilities of our method. Firstly, explicit cross-object generalization is only possible through leveraging the generalization capability of the visual entity detectors and subsequent retraining of the policy given the original demonstration. It would be interesting to investigate how we can **obtain a more general representation that allows for reusing a trained policy on different object classes without having train the policy from scratch**. Secondly, generalization across different configurations of the scene with the same object setup was only investigated by perturbing the initial starting configurations of all objects in the scene with subsequent retraining of the entire policy. Again, it would be interesting to investigate **generalization capabilities of the final policy, and also generalization across a more diverse set of initial starting configurations**. We currently hard-imitate the given demonstration, meaning we directly optimize for matching the demonstration state-trajectory without any regards for optimizing a global policy. One could employ some type of visual Guided Policy Search [33] to learn a globally valid policy, which is bootstrapped from multiple single imitation trajectories as discussed in this work.

With respect to the visual representation learning part, an interesting direction for future work would be to **propose object-specific parts rather than generic pixel features**. The agent could infer from multiple demonstrations and self-exploration what constitutes a relevant sub-part of an object, e.g., the handle and opening of a mug. This would facilitate generalization across different object-instances from the same class, and would further allow for a more sophisticated state space representation since these sub-parts would ideally be trackable, and thus, dynamically consistent over time. Or one could heuristically define a set of interpretable point features *a priori*.

Meta-learning of the edge weights and their binary attention is also an exciting extension of our current work. Using multiple demonstrations of the same task instead of just a single one would allow for inferring the weights rather than relying on heuristic motion attention of the binary attention. Furthermore, employing some type of intervention would allow for specifically learning causal relationships between the entities, encoded as the binary attention and the weights of the edges between them.

6.3 Lessons Learned

Learning Policies on Real Robotic Systems: One of the major challenges was policy training on the Baxter robot. Learning manipulation tasks on a real robot requires taking into account many more variables than working with a simulation. We spent a lot of time fine-tuning low-level control interfaces in terms of position and velocity control. Furthermore, the Baxter robot itself is only accurate up to 1cm precision with respect to its end-effector (X, Y, Z)-position, which made it hard to decorrelate a bad policy rollout from precision-based errors inherent to the system. For this reason, most learned tasks required repeated rollouts, and if the tasks involved grasping the object, the resetting process demanded tedious manual placement of all objects to their original place. Since trajectory optimization methods are especially dependent on starting from the same state for every rollout, this introduced extra noise into the learning process which was hard to account for on a systematic level. Since singular configurations were more likely at different location in task space depending on the task, this made it difficult to find one single scene configuration that allowed for a streamlined execution of experiments. For example, the pouring task required the end-effector's principle axis to point horizontally, while the stacking and pushing experiment required it to point downwards.

Furthermore, some variations of a task with different initial start positions of the objects used in the task led to the same issue, i.e. the robot got stuck close to a singular position, leading to repeated failure of inverse kinematic calculations. Furthermore, even though we stored all initial configurations of the right arm along with the mounted camera as well as the left arm which was used for manipulation, commanding the robot to those initial positions was never accurate due to Baxter only being precise up to roughly 1 cm. This further complicated the learning process. Overall, working with a real robotic system platform introduces a variety of potential noise and error sources that one has to take into consideration. This complicates the process of decorrelating the effects of all involved variables on the performance of the method. However, once the method works, this is a good sign for its robustness, while simulation performances often fail to deliver in real life due to the absence of such random noise variables.

We also note that for certain task setups, especially for the *pouring* task, we encountered difficulties in ensuring that the robot would not get too close to reaching joint limits during the execution. More specifically, propping the robot arm in a way that facilitated pouring motions led to a very restricted configuration in joint space. This reveals an interesting point for future investigation: how "good" are certain joint & null space configurations when taking into account future desired configurations, and how can we include this kind of reasoning into multi-step prediction? For example, if the robot has to change its orientation after terminating a sub-task, this orientation change might be much easier to execute if it had started off in a different initial configuration.

Robustness of Visual Detectors in Dynamic Real-Life Setups: We faced numerous challenges when training our visual detectors, both with respect to the Mask R-CNN and the pixel feature learning. Especially dealing with different lighting conditions that changed during the course of the day as well as our choice of using shiny cans as manipulation objects contributed to the extra challenge of ensuring robustness of our visual entity detectors. Only heavy augmentation and fine-tuning on problematic objects in a targeted manner allowed for robust detection. This shows that even a structured approach like ours, where we split object detection and reward computation in two separate pipelines struggles with robustness issues, but was at least indicative of failure cases due to the visual feedback we got from the actual detection masks. Now consider instead a setup of extracting deep features from the entire input image, where there is no such thing as a "failed" detection, but will always yield a feature vector regardless of disturbances in the scene. This might be disadvantageous in a sense that the neural architecture might be presented an input image that is perturbed, e.g. by different lighting, in such a way such that it is outside of the training distribution, but there is no simple way of indicating this deviation from the learned manifold other than tediously sanity-checking the output feature. We still have a long way to go to ensure robustness in deep learning setups, and especially in robotic settings where often humans are involved or the integrity of an object is at stake, we need to find effective ways of formally guaranteeing robustness.

Deep Feature Learning: We further experimented using deep features for object node representations, but found the learning process to be hard to debug, as mentioned in the previous chapter. More specifically, we trained a deep neural network as a pose predictor given the image crop of an object. While the pose predictor worked sufficiently well for viewpoints that did not deviate too much from the data encountered during training time, the pose prediction was prone to yield erroneous outputs for viewpoints outside of the training distribution. For this reason, we decided to change our approach to also resort to geometric feature points, in our case interpretable point features, to yield an explicit representation of the object. While this representation is still prone towards yielding poor results for unseen viewpoints, it at least shows more interpretability in this regard.

Appendix A

Appendix

A.1 Hyperparameters for Experimental Section

This is a comprehensive list of the hyperparameters used for our experiments. We note that this list might be non-exhaustive in the sense that we do not list hyperparameters which were copied over from their respective source implementation or if they are not detrimental to the performance our method to our knowledge. For example, we refrain from listing parameters such as the number of Anchor Scales used for training the Mask R-CNN architecture since this can be inferred from their source implementation.

***Pushing* Task:**

1. Action space: 4-dimensional (XYZ + rotation)
2. Time step length: 0.4 sec
3. Episode length: 30
4. Cost weight hand edges: 1.0 (50.0 for direction-change variant)
5. Cost weight object edges: 1.0
6. Cost weight point edges: 0.0 (not used)
7. Number of rollouts per update cycle: 8
8. Controller gains: XYZ: 0.001, rotation: 0.02

***Stacking* Task:**

1. Action space: 4-dimensional (XYZ + gripper state)
2. Time step length: 0.4 sec
3. Episode length: 30
4. Cost weight hand edges: 1.0
5. Cost weight object edges: 1.0
6. Cost weight point edges: 0.0 (not used)

7. Number of rollouts per update cycle: 8
8. Controller gains: XYZ: 0.001

***Pouring* Task:**

1. Action space: 4-dimensional (XYZ + rotation)
2. Time step length: 0.4 sec
3. Episode length: 20
4. Cost weight hand edges: 1.0
5. Cost weight object edges: 1.0
6. Cost weight point edges: 1.0
7. Number of rollouts per update cycle: 10
8. Controller gains: XYZ: 0.00125, rotation: 0.02

***Transportation* Task:**

1. Action space: 4-dimensional (XYZ + rotation)
2. Time step length: 0.4 sec
3. Episode length: 20
4. Cost weight hand edges: 1.0
5. Cost weight object edges: 1.0
6. Cost weight point edges: 1.0
7. Number of rollouts per update cycle: 10
8. Controller gains: XYZ: 0.001, rotation: 0.02

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *International Journal of Robotics Research*, 29(13): 1608–1639, 2010. ISSN 02783649. doi: 10.1177/0278364910371999. URL `http://ai.stanford.edu/{~}acoates/papers/AbbeelCoatesNg{_}IJRR2010.pdf`. 1

[2] Pulkit Agrawal, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Learning to poke by poking: Experiential learning of intuitive physics. *CoRR*, abs/1606.07419, 2016. URL `http://arxiv.org/abs/1606.07419`. 3.1

[3] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, June 2016. doi: 10.1109/CVPR.2016.110. 3.1, 3.2

[4] Smruti Amarjyoti. Deep reinforcement learning for robotic manipulation-the state of the art. *arXiv preprint arXiv:1701.08878*, 2017. 1.2

[5] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robot. Auton. Syst.*, 57(5):469–483, May 2009. ISSN 0921-8890. doi: 10.1016/j.robot.2008.10.024. URL `http://dx.doi.org/10.1016/j.robot.2008.10.024`. 3.1

[6] Michael Bain and C Sommut. A framework for hehavioural claning. *Machine Intelligence 15*, pages 103–129, 2001. doi: 10.1.1.25.1759. 2.4

[7] Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. *CoRR*, abs/1612.00222, 2016. URL `http://arxiv.org/abs/1612.00222`. 3.1, 3.2

[8] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinícius Flores Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, aglar Gülehre, Francis Song, Andrew J. Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey R. Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matthew Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks. *CoRR*, abs/1806.01261, 2018. 1.2, 6

[9] Aude Billard. Learning human arm movements by imitation: Evaluation of a biologically-inspired... — Request PDF. 941:1–16,

2001. URL https://www.researchgate.net/publication/2630634{_}Learning{_}human{_}arm{_}movements{_}by{_}imitation{_}Evaluat 1

[10] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning. 2017. ISSN 00308870. doi: 10.1016/j.aqpro.2013.07.003. URL http://arxiv.org/abs/1703.03078. 2.3.3, 3.1, 4.4

[11] Christopher B Choy, JunYoung Gwak, Silvio Savarese, and Manmohan Chandraker. Universal correspondence network. In *Advances in Neural Information Processing Systems*, pages 2414–2422, 2016. 4.2.3

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 1.2

[13] Coline Devin, Pieter Abbeel, Trevor Darrell, and Sergey Levine. Deep Object-Centric Representations for Generalizable Robot Learning. 2017. URL http://arxiv.org/abs/1708.04225. 3.2

[14] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 240–247, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390187. URL http://doi.acm.org/10.1145/1390156.1390187. 3.2

[15] C. Finn, M. Zhang, J. Fu, X. Tan, Z. McCarthy, E. Scharff, and S. Levine. Guided policy search code implementation, 2016. URL http://rll.berkeley.edu/gps. Software available from rll.berkeley.edu/gps. 4.5, 5.2

[16] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Learning visual feature spaces for robotic manipulation with deep spatial autoencoders. *CoRR*, abs/1509.06113, 2015. URL http://arxiv.org/abs/1509.06113. 3.1

[17] Peter Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *Conference on Robot Learning*, 2018. 4.2.3, 4.5

[18] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards. *CoRR*, abs/1511.07404, 2015. URL http://arxiv.org/abs/1511.07404. 3.1, 3.2

[19] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors. In *IEEE International Conference on Intelligent Robots and Systems*, volume 2016-November, 2016. ISBN 9781509037629. doi: 10.1109/IROS.2016.7759592. 2.3.2

[20] Golnaz Ghiasi, Yi Yang, Deva Ramanan, and Charless Fowlkes. Parsing occluded people. In *CVPR*, 2014. 3.1

[21] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870. (document),

2.1, 2.2, 4.2.1, 4.2.1, 4.5

[22] Nicolas W. Hengartner. A note on maximum likelihood estimation. *American Statistician*, 53(2):123–125, 1999. ISSN 15372731. doi: 10.1080/00031305.1999.10474444. URL `https://homes.cs.washington.edu/{~}todorov/papers/TassaICRA14.pdf`. 2.3

[23] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4565–4573. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf`. 3.1

[24] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9370(2010):84–92, 2015. ISSN 16113349. doi: 10.1007/978-3-319-24261-3_7. 5

[25] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2):21:1–21:35, April 2017. ISSN 0360-0300. doi: 10.1145/3054912. URL `http://doi.acm.org/10.1145/3054912`. 3.1

[26] Ashesh Jain, Amir Roshan Zamir, Silvio Savarese, and Ashutosh Saxena. Structural-rnn: Deep learning on spatio-temporal graphs. *CoRR*, abs/1511.05298, 2015. URL `http://arxiv.org/abs/1511.05298`. 3.1, 3.2

[27] Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lázaro-Gredilla, Xinghua Lou, Nimrod Dorfman, Szymon Sidor, D. Scott Phoenix, and Dileep George. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *CoRR*, abs/1706.04317, 2017. 3.2

[28] Anna Khoreva, Rodrigo Benenson, Eddy Ilg, Thomas Brox, and Bernt Schiele. Lucid data dreaming for object tracking. *CoRR*, abs/1703.09554, 2017. URL `http://arxiv.org/abs/1703.09554`. 3.1

[29] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, ICRA'09, pages 2509–2515, Piscataway, NJ, USA, 2009. IEEE Press. ISBN 978-1-4244-2788-8. URL `http://dl.acm.org/citation.cfm?id=1703775.1703856`. 1

[30] Oliver Kroemer and Gaurav S. Sukhatme. Learning relevant features for manipulation skills using meta-level priors. *CoRR*, abs/1605.04439, 2016. URL `http://arxiv.org/abs/1605.04439`. 1.2

[31] Kyu Hwa Lee, Jinhan Lee, Andrea L. Thomaz, and Aaron F. Bobick. Effective robot task learning by focusing on task-relevant objects. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009*, pages 2551–2556, 2009. doi: 10.1109/IROS.2009.5353979. 4.3

[32] Timothée Lesort, Natalia Díaz Rodríguez, Jean-François Goudou, and David Filliat. State

representation learning for control: An overview. *CoRR*, abs/1802.04181, 2018. URL `http://arxiv.org/abs/1802.04181`. 3.1

[33] Sergey Levine and Pieter Abbeel. Learning Dynamic Manipulation Skills under Unknown Dynamics with Guided Policy Search. *Advances in Neural Information Processing Systems 27*, pages 1–3, 2014. ISSN 10504729. doi: 10.1109/ICRA.2015.7138994. URL `http://papers.nips.cc/paper/5444-learning-neural-network-policies-with-guided-policy-search-under-u` `pdf`. 2.3.2, 2.3.2, 6.2

[34] Shuang Li, Xiaojian Ma, Hongzhuo Liang, Michael Görner, Philipp Ruppel, Bin Fang, Fuchun Sun, and Jianwei Zhang. Vision-based Teleoperation of Shadow Dexterous Hand using End-to-End Deep Neural Network. URL `https://arxiv.org/pdf/1809.06268.pdf`. 4.2.2

[35] Weiwei Li and Emanuel Todorov. Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems. pages 222–229, 2011. doi: 10.5220/0001143902220229. URL `https://homes.cs.washington.edu/{~}todorov/papers/LiICINCO04.pdf`. 2.3.1

[36] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8693 LNCS(PART 5):740–755, 2014. ISSN 16113349. doi: 10.1007/978-3-319-10602-1_48. 4.2

[37] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014. URL `http://arxiv.org/abs/1405.0312`. 1.2

[38] YuXuan Liu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation. (Nips), 2017. URL `http://arxiv.org/abs/1707.03374`. 2.4, 3.1

[39] Lucas Manuelli, Wei Gao, Peter R. Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *CoRR*, abs/1903.06684, 2019. URL `http://arxiv.org/abs/1903.06684`. 4.2.3

[40] By DAVID MAYNE. A Second-order Gradient Method for Determining Optimal Trajectories of Non-linear Discrete-time Systems. *International Journal of Control*, 3(1):85–95, 1966. doi: 10.1080/00207176608921369. URL `https://doi.org/10.1080/00207176608921369`. 2.3.1

[41] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL `http://arxiv.org/abs/1312.5602`. 1.2

[42] K. Mülling, J. Kober, O. Kroemer, and J. Peters. Learning to select and generalize striking movements in robot table tennis. *International Journal of Robotics Research*, 32(3):263–279, 2013. 3.1

[43] Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. *CoRR*, abs/1703.02018, 2017. URL http://arxiv.org/abs/1703.02018. 5

[44] Chrystopher L. Nehaniv and Kerstin Dautenhahn. Imitation in animals and artifacts. chapter The Correspondence Problem, pages 41–61. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-04203-7. URL http://dl.acm.org/citation.cfm?id=762896.762899. 3.1

[45] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1-55860-707-2. URL http://dl.acm.org/citation.cfm?id=645529.657801. 3.1

[46] Dennis Park and Deva Ramanan. Articulated pose estimation with tiny synthetic videos. *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 58–66, 2015. 3.1

[47] Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A. Efros, and Trevor Darrell. Zero-shot visual imitation. In *ICLR*, 2018. 3.1

[48] Morgan Quigley, Ken Conley, Brian P Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, volume 3, 2009. 4.5

[49] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *CoRR*, abs/1709.10087, 2017. URL http://arxiv.org/abs/1709.10087. 3.1

[50] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. ISSN 01628828. doi: 10.1109/TPAMI.2016.2577031. 2.1

[51] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. DL-04-2_2017_Faster R-CNN(IEEEE). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017. ISSN 01628828. doi: 10.1109/TPAMI.2016.2577031. URL https://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-propos
pdf. 4.2.1

[52] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "GrabCut": interactive foreground extraction using iterated graph cuts. *Acm Tg*, 23(3):309, 2004. ISSN 07300301. doi: 10.1145/1015706.1015720. URL https://cvg.ethz.ch/teaching/cvl/2012/grabcut-siggraph04.pdf. 4.2.1

[53] Fereshteh Sadeghi and Sergey Levine. (cad)$ˆ2$rl: Real single-image flight without a

single real image. *CoRR*, abs/1611.04201, 2016. URL http://arxiv.org/abs/1611.04201. 3.1

[54] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018. 3.2

[55] S. Schaal. Is imitation learning the route to humanoid robots? 3(6):233–242, 1999. URL http://www-clmc.usc.edu/publications/S/schaal-TICS1999.pdf; http://www-clmc.usc.edu/publications/S/schaal-TICS1999-rep.pdf. 3.1

[56] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters*, 2(2):420–427, 2017. 4.2.3

[57] Pierre Sermanet, Corey Lynch, Jasmine Hsu, and Sergey Levine. Time-Contrastive Networks: Self-Supervised Learning from Multi-view Observation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017-July:486–487, 2017. ISSN 21607516. doi: 10.1109/CVPRW.2017.69. (document), 1.2, 1.2, 3.1, 5, 5, 5.1, 5.3

[58] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017. 4.2.2

[59] Bradly C. Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *CoRR*, abs/1703.01703, 2017. URL http://arxiv.org/abs/1703.01703. 3.1

[60] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction*. Adaptive computation and machine learning. MIT Press, 1998. ISBN 0262193981. URL http://www.worldcat.org/oclc/37293240. 2.2

[61] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. 2015. ISSN 08866236. doi: 10.1109/CVPR.2016.308. URL http://arxiv.org/abs/1512.00567. 5

[62] Evangelos Theodorou, Jonas Buchli, and Stefan Schaal. Learning policy improvements with path integrals. *Artificial Intelligence and Statistics*, pages 828–835, 2007. ISSN 15324435. URL http://www.scopus.com/inward/record.url?eid=2-s2.0-84862297667{&}partnerID=tZOtx3y1. 2.3.3

[63] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907. 3.1

[64] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *IJCAI International Joint Conference on Artificial Intelligence*, 2018-July:4950–4957, 2018. ISSN 10450823. doi: 10.24963/ijcai.2018/687. URL https://www.ijcai.org/proceedings/2018/0687.pdf. 3.1

[65] Manuel Watter, Jost Tobias Springenberg, Joschka Boedecker, and Martin A. Riedmiller.

Embed to control: A locally linear latent dynamics model for control from raw images. *CoRR*, abs/1506.07365, 2015. URL `http://arxiv.org/abs/1506.07365`. 3.1

[66] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation. 2017. doi: 10.1109/ICRA.2018.8461249. URL `http://arxiv.org/abs/1710.04615`. 3.1

[67] Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. *CoRR*, abs/1710.04615, 2017. URL `http://arxiv.org/abs/1710.04615`. 3.1